all rights reversed

HACKER

Απρίλης 2014 # Τιμή τεύχους 4€

τεύχος 031 - True Randomness Edition

Το εξαιρετικά ύπουλο Heartbleed bug [*]



Από το Minecraft στο Raspberry Pi

Σπιτικό cloud storage κι αντίο Dropbox

Η ευλογία της πραγματικής τυχαιότητας

Το βολικό WebStorage και μια ύπουλη αδυναμία

[*] Επεξήγηση κι ανάλυση, όχι μόνο περιγραφές!

all rights reversed

HACKER

Περιεχόμενα

και οδηγίες χρήσης

Εκδότης / Διευθυντής	Χρήστος Βαρελάς
Αρχισυντάκτης	Παναγιώτης Βαρελάς
Σύμβουλοι έκδοσης	Νίκος Μουρατίδης, Βούλα Παυλίδου
Σύνταξη	Γιώργος Αραβίδης, Παναγιώτης Βαρελάς, Χρήστος Βαρελάς, Ανδρέας Βενιέρης, Γιώργος Γιάννου, Μανώλης Κιαγιάς, Ιωάννης Κονιάρης, Πέτρος Κυλαδίτης, Γιώργος Μελέκος, Νίκος Μουρατίδης, Νικόλαος Σηφάκης, Μέσος Παπαδόπουλος (TM), Χρίστος Τόμπρας
Γραμματική επιμέλεια	Βούλα Παυλίδου
Καλλιτεχνική Επιμέλεια	Πέτρος Φιλιππίδης

Το ηλεκτρονικό περιοδικό deltaHacker είναι συνδρομητικό και εκδίδεται από την Parabing Creations δώδεκα φορές το χρόνο, κάθε μήνα. Η Parabing Creations έχει την έδρα της στη Μαιώτιδος 3, 55133, Καλαμαριά. Ο δικτυακός τόπος του περιοδικού είναι στο http://deltahacker.gr. Το email επικοινωνίας είναι το talk2us@deltahacker.gr και ειδικά για τις συνδρομές είναι το subscriptions@deltahacker.gr. Οι γενικές πληροφορίες για τις συνδρομές παρατίθενται στο http://deltahacker.gr/subscriptions και οι ίδιες οι παραγγελίες γίνονται από το http://deltahacker.gr/pdf12order. Όλες οι απόψεις που εκφράζονται στα άρθρα δεν εκφράζουν απαραίτητα και τη γνώμη του περιοδικού.

Της ατάκας και του πληκτρολογίου

Αγαπητέ εαυτέ,

- Προσπάθησε να μην παρεκτραπείς (και) σε αυτό το editorial.
- Όχι ότι γράφεις συχνά το κείμενο που καλωσορίζει τους αναγνώστες στο εκάστοτε τεύχος.
- Όταν το κάνεις, όμως, συνηθίζεις να βγάζεις όλο σου το μένος προς κάποια ομάδα ανθρώπων.
- Αυτό δεν σε καθιστά "ρατσιστή", όπως μπορεί να σκεφτούν κάποιοι multi-culti-seize-the-moment-make-it-count τύποι, που συνηθίζουν να συγχέουν ακόμα και τις πιο απλές έννοιες, αλλά, τέλος πάντων, σε κάνει να μοιάζεις με κάποιον παράφρονα που μόλις τον αμόλησαν ελεύθερο.
- Και τώρα σταμάτα να γράφεις σε τρίτο πρόσωπο για τον εαυτό σου, γιατί θα σε περάσουν στ’ αλήθεια για τρελό...

Αγαπητές φίλες, αγαπητοί φίλοι,

- Καλό μήνα να έχουμε και καλό κουράγιο σε όλους! Έχουμε εκλογές σε λίγες μέρες και θα χρειαστούμε αρκετή αντοχή και υπομονή.
- Κάποιοι θα χρειαστούν αντοχή για να ολοκληρώσουν την προεκλογική τους καμπάνια.
- Κάποιοι άλλοι θα τη χρειαστούν για να δώσουν με τους καλύτερους δυνατούς όρους *και* αυτή την πολιτική μάχη.
- Μια άλλη ομάδα, που δυστυχώς απαριθμεί αρκετά μέλη, θα χρειαστεί αντοχή για να ολοκληρώσει την καμπάνια της στο Twitter και στο Facebook.
- Οι υπόλοιποι θα χρειαστούν αντοχή και κυρίως υπομονή, για να υποστούν την καμπάνια των τελευταίων στα social media.
- Έχω κι εγώ τις πολιτικές μου απόψεις και θα ήταν ανόητο να το κρύψω. Μολαταύτα, δεν τα έχω βάλει με τους υποψηφίους των *άλλων* πολιτικών δυνάμεων.
- Αυτοί κάνουν τη δουλειά τους. Μπορεί με κάποιους να διαφωνώ, με κάποιους άλλους να βγάζω καντήλες και για κάποιους τρίτους να εύχομαι την απομάκρυνσή τους... από το συγκεκριμένο παράλληλο σύμπαν, γενικά. Ας πάνε δηλαδή σε κάποιο άλλο, τούτο δεν είναι το μοναδικό ωραίο κι ενδιαφέρον.
- Αλλά ακόμη και να μείνουν, οφείλω να αναγνωρίσω σε όλους το γεγονός ότι ζουν και λειτουργούν μέσα στον κόσμο.
- Κι αυτό είναι πολύ σημαντικό, φίλες και φίλοι. Είτε μας αρέσει ο κόσμος και θέλουμε να τον διατηρήσουμε αναλλοίωτο, είτε δεν μας αρέσει και θέλουμε να τον αλλάξουμε, πρέπει να επικεντρώνουμε τις δυνάμεις μας ακριβώς εκεί: Μέσα στον κόσμο.

- Αυτοί που καταφέρνουν να εξαντλήσουν την υπομονή μου τις τελευταίες μέρες, είναι όσοι εχθρεύονται *όλους* τους υποψηφίους γενικά κι αόριστα κι από το πρωί μέχρι το βράδυ εξαπολύουν πύρινα... tweets.
- Τι είναι αυτό που σας ενοχλεί, βρε παλικάρια και μορφονιές; Σας ενοχλούν οι απόψεις αυτές καθαυτές, που εκφράζουν ορισμένοι υποψήφιοι; Ή μήπως το γεγονός ότι κάποιοι άνθρωποι αποφάσισαν να καταναλώσουν ενέργεια για τη διάδοση μιας άποψης;
- Στην πρώτη περίπτωση, θα μπορούσατε απλά να μας πείτε με ποιες απόψεις διαφωνείτε και, τέλος πάντων, ποιες είναι οι δικές σας. Όχι ότι σώνει και καλά μας ενδιαφέρει, αλλά αυτό θα ήταν το λογικό. Σε κάποιους από εμάς, πάντως, καρφί δεν μας καίγεται -- και νομίζω ότι δεν είναι και τόσο τίμιο να το κρύβουμε.
- Στη δεύτερη περίπτωση, αν αυτό που σας ενοχλεί είναι το γεγονός ότι κάποιοι άνθρωποι αποφασίζουν να ασχοληθούν με την πολιτική, πρέπει να καταλάβετε το εξής: Το γεγονός ότι εσείς κατακεραυνώνετε τους πάντες, με μανία και κυρίως μέσα από το Internet, δεν σημαίνει ότι απέχετε από την πολιτική. Να ξέρετε ότι συμμετέχετε *και* εσείς ενεργά, και μάλιστα αναπαράγετε τις κυρίαρχες αντιλήψεις: Αυτές που θέλουν τον κόσμο θεατή και στην καλύτερη περίπτωση ανεγκέφαλο χούλιγκαν, που εκφράζει τη δυσαρέσκειά του μέσω της αποχής.
- Τώρα, βέβαια, θα μου πείτε ότι υπάρχουν και υποψήφιοι που πολιτεύονται αποκλειστικά εξαιτίας της ματαιοδοξίας τους. Ίσως αυτοί να ερεθίζουν τη λογική, την ηθική και την ατόφια επαναστατική διάθεση των ακτιβιστών του πληκτρολογίου...
- Κατ' αρχάς, πρέπει να ξεκαθαρίσουμε το εξής: Είτε θέτεις υποψηφιότητα επειδή είσαι ψώνιο, είτε κάνεις επανάσταση μέσω twitter, ένα είναι το σίγουρο: Ζεις στον κόσμο σου.
- Χρειάζεται να πω κι άλλα;
- Τι έπαθες παιδάκι μου κι αποφάσισες να γεμίσεις το timeline μου με βρισιές για τις τράπεζες, τους απεργοσπάστες και τον καπιταλισμό; Τι σου συνέβη αγόρι μου και ξεσπαθώνεις στο πληκτρολόγιο; Έχασε η ομάδα ή σε παράτησε η φίλη σου;
- Ούτε και σε μένα αρέσει η απεργοσπασία, αλλά για να γίνει κανείς απεργοσπάστης, πρέπει να είναι εργαζόμενος. Όπως και να το κάνουμε, ο εφοπλιστής δεν μπορεί να γίνει απεργοσπάστης. Εφόσον μιλάμε για κάποιον εργαζόμενο, λοιπόν, θα φροντίσω να τον προσεγγίσω και να του μιλήσω σαν άνθρωπος. Αν διαπιστώσω ότι δεν έχει νόημα, τότε πολύ απλά θα τον αγνοήσω. Τα δικτυακά μαθήματα εργατικής ηθικής δεν σε κάνουν να φαίνεσαι cool. Ξέρω ότι αυτό θα ήθελες, αλλά κανένας δεν σε βλέπει έτσι.
- Και με τον καπιταλισμό διαφωνώ. Τον θεωρώ εκμεταλλευτικό σύστημα και, τέλος πάντων, ξεπερασμένο για το σημερινό επίπεδο ανάπτυξης των παραγωγικών δυνάμεων. Αυτό όμως δεν αποτελεί λόγο για να πρήξω τον κόσμο που με "ακολουθεί" στα social media. Όλα αυτά θα τα συζητήσω με τους συναδέλφους μου, με τους φίλους και με τους γνωστούς. Οι επαφές στο Twitter και στο Facebook δεν είναι τίποτα απ' όλα αυτά. Κι αν κάποιος εκεί μέσα είναι *και* συνάδελφος ή φίλος, θα συζητήσω μαζί του από κοντά...

- Ξέρεις γιατί; Γιατί πολύ απλά, όταν συζητάω αυτά τα πράγματα, θέλω να πείθω το συνομιλητή μου και όχι να ξεδίνω, ούτε να παριστάνω τον ξερόλα.
- Ουφ! Είχα δεν είχα, συγχύστηκα πάλι. Αλλά γι' αυτό δεν μπορείτε να είστε σίγουροι, οπότε το προσπερνώ γρήγορα γρήγορα και συνεχίζω.
- Για την όποια σύγχυση, πάντως, μη νομίζετε ότι οφείλεται αποκλειστικά στην προεκλογική περίοδο. Οι επαναστάτες της ατάκας δεν αποτελούν καινούργιο φαινόμενο. Στοιχειώνουν το Διαδίκτυο χρόνια τώρα, μεταξύ άλλων αναζητώντας "ξεπουλημένους", "προδότες" και "ανίκανους" για να τους κατακεραυνώσουν. Και μην τολμήσεις να κάνεις κάτι με τρόπο που δεν εγκρίνουν... Εάν υποψιαστούν ότι δεν είσαι τόσο ικανός χρήστης της τεχνολογίας όσο εκείνοι, δεν σε γλυτώνει τίποτα. Είσαι ένας άχρηστος κι οτιδήποτε έχεις να πεις είναι σίγουρα ανοησία... (Σ.τ.Ε. Απίδια ικανοί χρήστες της τεχνολογίας είναι κάποιοι εξ αυτών, αλλά τέλος πάντων.)
- Κάτι τέτοιες αντιλήψεις, φίλοι μου, είναι που εξοργίζουν τον χάκερ που κρύβω μέσα μου. Και μπορεί αυτός ο χάκερ να μην είναι μεγάλος και τρανός --στην πραγματικότητα περί χακερονίου πρόκειται--, αλλά θέλω να πιστεύω ότι είναι τόσο λογικός και τόσο έξυπνος, όσο χρειάζεται για να αντιλαμβάνεται την ευφυΐα που κουβαλάει ο καθένας μας ;)
- Κι όταν μπορεί κανείς να εντοπίσει την ευφυΐα όσων βρίσκονται γύρω του, όχι μόνο δεν είναι υπερόπτης, αλλά του είναι και ανυπόφορη η συναναστροφή με οποιονδήποτε εξυπνάκια...
- Ειδικά με εκείνους που κηρύσσουν την επανάσταση από το πληκτρολόγιο, φοράνε κουκούλα και νομίζουν ότι το life-style του "urban rebel" τρομοκρατεί αυτούς που μας έχουν κατακλέψει.
- Μην τους κάνετε τη χάρη, λοιπόν. Μπορεί να μην έχουμε ανάγκη από εκλογές, αλλά αφού γίνονται, πρέπει να πάρουμε θέση.
- Καλή ψήφο λοιπόν κι απολαύστε το -- ό,τι κι αν ψηφίσετε!
- Εντάξει, σχεδόν "ό,τι κι αν ψηφίσετε".

Σας χαιρετώ και με τη νίκη,

Ο κος Αρχισυντάκτης™



σνέιλ μέιλ

Αγαπητοί μου,

Μεγάλη Πέμπτη, σήμερα, κι ετοιμαζόμουν να ξεκινήσω τη φετινή μου πασχαλιάτικη επιστολή με τις παραδοσιακές ευχές της εποχής, αλλά με πρόλαβε η εξαδέλφη.

"Α, γράφεις;" μου λέει. "Σου πέρασε το writer's block;"

Την κοίταξα απορημένος.

"Τελικά δεν κατάλαβα", συνέχισε. "Τι ήταν αυτό που έγραφες την περασμένη φορά, για τις ανάμικτες καραμέλες; Είχες κάτι κατά νου, ή ήταν το writer's block;"

"Ποιο writer's block, τι εννοείς;" διαμαρτυρήθηκα εγώ. "Δεν είχα κανένα writer's block. Ματιασμένος ήμουνα".

"Καλά, σοβαρός άνθρωπος, το βρίσκεις αυτό σοβαρή δικαιολογία;"

"Ακόμα ματιασμένος είσαι;" ακούστηκε ανήσυχη η φωνή της μητέρας της εξαδέλφης από την κουζίνα, όπου ετοιμαζόταν να βάψει τα αβγά. "Κάτσε, σε λίγο έρχεται η κυρία Καίτη."

"Αφήστε τον καλέ μαμά" την αποπήρε η εξαδέλφη. "Σιγά μην είναι ματιασμένος. Δικαιολογίες ψάχνει να μη μας απαντήσει."

"Λοιπόν, εντάξει", είπα εγώ. "Δεν είμαι ματιασμένος. Τι άλλο θέλεις να σου πω;"

"Για τις ανάμικτες καραμέλες. Να μου πεις τι εννοούσες."

"Κοίτα, ΟΚ. Να σου πω. Αυτό που μου είχε εμένα φανεί ενδιαφέρον στην ιστορία με τις ανάμικτες καραμέλες ήταν το πώς μια λέξη απλή και αυτονόητη, ας πούμε η λέξη 'ανάμικτος', μπορεί τόσο εύκολα να καταλήξει κενή νοήματος, αν της αλλάξεις πεδίο αναφοράς."

"Εντάξει αυτό το κατάλαβα. Γιατί μας το έλεγες, δεν κατάλαβα."

"Κοίτα, το μόνο που ήθελα να επισημάνω --και για να είμαστε ακριβείς, όχι εγώ: ο Χάιζενμπεργκ που αναφέρει την ιστορία στο βιβλίο του 'Φυσική και Φιλοσοφία'-- είναι πως ο όρος 'ανάμικτες', όταν αναφέρεται σε καραμέλες, εμπεριέχει μιαν ασάφεια. Από πόσες καραμέλες και πάνω μπορούμε να λέμε ότι είναι 'ανάμικτες'; Αυτό είναι το θέμα. Χρησιμοποιούμε λέξεις χωρίς να θυμόμαστε το πεδίο αναφοράς τους, και έτσι, αναπόφευκτα, καταλήγουμε σε ασάφειες, αντιφάσεις, και εννοιολογικά αδιέξοδα."

"Αυτό είναι όλο; Τώρα αυτό δεν το ξέραμε; Εξυπνάδες!" κάγχασε η εξαδέλφη.

"Μια που είσαστε εδώ και τα ξέρετε όλα", ακούστηκε ξανά από την κουζίνα η μητέρα της εξαδέλφης που είναι ινδουίστρια, "πέστε μου, τι χρώμα τελικά να τα κάνω τα αβγά; Εχετε καμιά προτίμηση; Εχω κόκκινο, πορτοκαλί, πράσινο..."

"Μην τον ρωτάτε αυτόν, θα σας πει να τα βάψετε ανάμικτα" είπε η εξαδέλφη και σηκώθηκε. "Έρχομαι να βοηθήσω".

Τον σκεφτόμουνα, που λέτε, αυτές τις μέρες τον Χάιζενμπεργκ, αγαπητοί μου, γιατί διάβαζα ένα πρόσφατο βιβλίο του Στήβεν Χώκινγκ, που μου έστειλε για το Πάσχα ο ανιψιός. Καμιά σχέση με Πάσχα, βέβαια, αλλά τέλος πάντων. "The Grand Design".

Θυμόμουν τον Χάιζενμπεργκ και τις ανάμικτες καραμέλες του, που τις έφερνε για παράδειγμα του πόσο ανεπαρκείς είναι οι έννοιες του δικού μας κόσμου όταν προσπαθούμε να τις χρησιμοποιήσουμε στον κόσμο της κβαντικής φυσικής.

Με τον ίδιο τρόπο που δεν μπορούμε να λέμε για ανάμικτες καραμέλες όταν έχουμε μόνο μία ή δύο, έλεγε ο Χάιζενμπεργκ, έτσι δεν μπορούμε να μιλάμε και για τον δυισμό κύμα / σωματίδιο. Δεν γίνεται κάτι να είναι και σωματίδιο και κύμα. Δεν είναι μόνο ότι μας φαίνεται παράλογο, σχολίαζε. Είναι παράλογο.

Βέβαια με την μαθηματική περιγραφή του ζητήματος δεν έχουμε πρόβλημα. Τις εξισώσεις τις κβαντομηχανικής τις λύνουμε μια χαρά. Το πρόβλημα έγκειται στην εποπτεία του πράγματος. Οι έννοιες "σωματίδιο" και "κύμα" είναι ανεπαρκείς όταν μιλάμε για κβαντικά φαινόμενα. Χρειαζόμαστε καινούργιες.

Αντίστοιχο θεωρούσε το πρόβλημα της ταχύτητας και της θέσης ενός σωματιδίου. Σύμφωνα με την αρχή της απροσδιοριστίας (την οποία αυτός διατύπωσε το 1927) δεν μπορούμε να γνωρίζουμε ταυτόχρονα και με ακρίβεια και τα δύο. Το πρόβλημα είναι το ίδιο, έγραφε ο Χάιζενμπεργκ. Πρόκειται για εννοιολογική ανεπάρκεια των εννοιών "θέση" και "ταχύτητα".

Θυμόμουν τον Χάιζενμπεργκ, που λέτε, και θυμόμουν την σεμνότητά του.

Το ακριβώς αντίθετο με τον Χώκινγκ.

Διαβάζω, λοιπόν, το "The Grand Design", και τον βλέπω (τον Χώκινγκ) να γράφει σελίδες επί σελίδων προσπαθώντας να απαντήσει στα μεγάλα ερωτήματα που ταλάνιζαν από πάντα την ανθρώπινη σκέψη:

Γιατί να υπάρχει κάτι και όχι το τίποτα; Γιατί υπάρχουμε; Γιατί έχει το σύμπαν τη μορφή που έχει;

Και πάει λέγοντας. You get the idea. (Παρεμπιπτόντως, στο πρώτο από τα ερωτήματα αυτά είχε αναφερθεί κάποια στιγμή και ο Κύριος Εκδότης. Περισσότερα στο http://deltahacker.gr/?p 7066.)

Βρίσκω την απάντηση του Χώκινγκ στην σελίδα 180, από όπου και μεταφράζω πρόχειρα:



"Επειδή λοιπόν υπάρχει ένας νόμος όπως αυτός της βαρύτητας, το Σύμπαν θα μπορούσε να έχει, και πράγματι έχει, δημιουργηθεί από μόνο του, από το μηδέν (...). Ο λόγος που υπάρχει κάτι και όχι το τίποτα είναι η αυθόρμητη δημιουργία spontaneous creation · γι' αυτό υπάρχει το σύμπαν· γι' αυτό υπάρχουμε κι εμείς οι ίδιοι. Δεν χρειάζεται να επικαλούμαστε κάποιον Θεό που τάχα μου θ' ανάψει το φως και θα θέσει το σύμπαν σε κίνηση."

Να την, αγαπητοί μου, η ανάμικτη καραμέλα. Τη βλέπετε;

Βρίσκεται στην ερώτηση "γιατί;"

Ο Χώκινγκ ρωτάει κάτι ("Γιατί υπάρχει κάτι και όχι το τίποτα;"), δίνει μιαν απάντηση ("Ο λόγος είναι η αυθόρμητη δημιουργία κ.λπ. κ.λπ."), και νομίζει ότι έτσι έχει απαντήσει στα μεγάλα ερωτήματα που ταλάνιζαν από πάντα την ανθρώπινη σκέψη.

Κάνει λάθος.

Γιατί χρησιμοποιεί τη λέξη "γιατί" με δύο διαφορετικούς τρόπους και δεν αντιλαμβάνεται ότι της αλλάζει το πεδίο αναφοράς.

Ας δούμε πώς. Ας πούμε ότι βλέπω την εξαδέλφη να πίνει κοκακόλα. Τη ρωτάω: "Γιατί πίνεις κοκακόλα;"

Και αυτή μου απαντάει.

"Επειδή διψούσα" / "Δεν είχε τίποτα άλλο να πάρω" / "Την άφησε η Μαιρούλα και την πίνω για να μην πάει χαμένη..." / "Γιατί έτσι" / "Ωχ, κοκακόλα μου φέρανε; Πέπσι ήθελα" / "Κάνω ένα συγκριτικό τεστ αναψυκτικών και τα δοκιμάζω όλα ένα προς ένα" / "Μου αρέσει" / "Νηστεύω" / "Έχω τα γενέθλιά μου" / "Είχα τα γενέθλιά μου" / κ.λπ. κ.λπ. κ.λπ.

Ποια από όλες αυτές τις απαντήσεις είναι έγκυρη; Όλες.

Ποια είναι η σωστή; Δεν ξέρω. Όλες τους, όμως, αναφέρονται στην εικαζόμενη προθετικότητα μιας πράξης (της εξαδέλφης, εν προκειμένω, που πίνει κοκακόλα).

Η λέξη κλειδί εδώ είναι, φυσικά, η λέξη "προθετικότητα" (μιας πράξης). Η ερώτηση, έτσι όπως την έθεσα, αναφέρεται σε μια προθετικότητα.

Ας πούμε τώρα ότι είμαι ο Νεύτωνας και βλέπω το μήλο να πέφτει από την μηλιά. Ρωτάω. "Γιατί έπεσε το μήλο;"

Οι απαντήσεις που πιθανώς θα δώσω έχουν αναγκαστικά να κάνουν με τη βαρύτητα. Ξέρω πως το μήλο δεν έχει την πολυτέλεια να αποφασίσει αν θα πέσει ή όχι. Η ερώτηση "γιατί", όταν αναρωτιέμαι για το φυσικό φαινόμενο της πτώσης, δεν αναφέρεται σε προθετικότητες· αναφέρεται στους φυσικούς νόμους που (εικάζω πως) ισχύουν --αναφέρεται, δηλαδή, σε κανονικότητες.

Εδώ έγκειται η ανάμικτη καραμέλα. Ο Χώκινγκ παίρνει μιαν ερώτηση που αναφέρεται σε μια (εικαζόμενη) προθετικότητα ("Γιατί υπάρχει το σύμπαν;") και δίνει μιαν απάντηση που αναφέρεται σε μια (εικαζόμενη) κανονικότητα ("Το σύμπαν, στην τάδε ή τη δείνα του εκδοχή, εμφανίζεται ως προϊόν μιας διαδικασίας που την ονομάζουμε αυθόρμητη δημιουργία").

Αυτό είναι το λογικό σφάλμα.

Σαν να με ρωτάει ο ανιψιός "Γιατί είναι κόκκινα τα αβγά;" κι εγώ, αντί να τον αποπάρω που τόσα χρόνια εκεί στα ξένα, έχει αρχίσει να ξεχνάει τα ιερά και όσια του λαού μας, να του απαντάω πως "τα αβγά είναι κόκκινα επειδή η χρωστική με την οποία τα έχουμε περάσει απορροφάει όλα τα άλλα χρώματα και ανακλά το κόκκινο".

Λοιπόν. Αυτά τα ωραία. Πάω κι εγώ στην κουζίνα να τις βοηθήσω με τα αβγά γιατί αυτές είναι ικανές για όλα.

Σας ασπάζομαι,

Θείος Ακάκιος

YOUR VERY OWN PLAYGROUND IN THE CLOUD. SECURE & GREEN. PAY AS YOU GO.

Keep your data in Iceland and take advantage of GreenQloud's cloud services powered by abundant, stable and cost effective, clean energy

www.greenqloud.com

ComputeQloud™
Host & Manage

- Up to 16 CPUs and 32gb RAM in the KVM hypervisor
- 64-bit server image templates including Ubuntu, Debian, FreeBSD, CentOS and more
- Built-in firewall
- In-browser noVNC client and full root access via ssh
- Network separation at the hardware level
- Dedicated IP address and reserved bandwidth
- EC2 compatible
- Instant scalability
- Online dashboard

StorageQloud™
Backup & Share

- Triple-redundant storage
- Full encryption
- Sync across all platforms
- S3 compatible
- Bucket websites
- Automatic load-balancing



For more information visit our webiste: www.greenqloud.com

Skill: Intermediate
Tags: Sync, BitTorrent, Raspberry Pi, encryption, ownCloud

RaspOwnSync

by multiPetros

Σπιτικό Cloud Storage

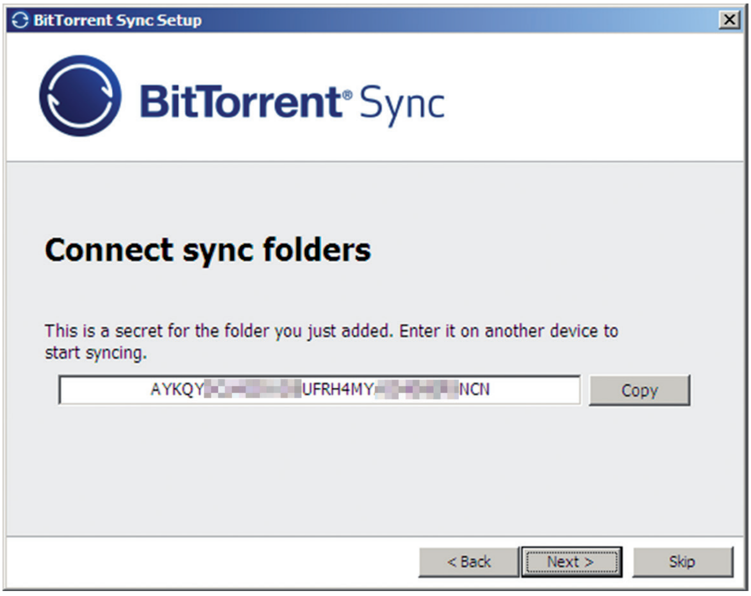
Αλήθεια, ποιος είναι αυτός που δεν προτιμάει τα σπιτικά κουλουράκια, σαν αυτά που φτιάχνουν το Πάσχα στα σπίτια, έναντι των βιομηχανοποιημένων; Άλλοι τα προτιμούν γιατί τους αρέσουν περισσότερο, άλλοι γιατί ξέρουν ακριβώς τι περιέχουν, άλλοι απλά γιατί τους αρέσει να τα φτιάχνουν οι ίδιοι. Η αλήθεια είναι ότι συμμεριζόμαστε και τις τρεις αυτές απόψεις. Γιατί λοιπόν να εναποθέτουμε τα προσωπικά μας αρχεία στις υπηρεσίες cloud storage τρίτων, και να μην έχουμε το δικό μας, σπιτικό cloud;

Οι υπηρεσίες cloud storage, πανταχού παρούσες και συνεχώς εξελισσόμενες, έχουν γίνει αναπόσπαστο κομμάτι του Internet στις μέρες μας. Στην πλειονότητα των περιπτώσεων, τις web υπηρεσίες συνοδεύει κι ένας desktop client, ο οποίος αναλαμβάνει τον αυτόματο συγχρονισμό των τοπικών αρχείων μ' αυτών που βρίσκονται στο cloud. Η χρηστικότητά του και η ευκολία που προσφέρει είναι αδιαμφισβήτητη, το αντίτιμο όμως δεν είναι ευκαταφρόνητο. Φυσικά, δεν αναφερόμαστε στο χρηματικό σκέλος. Εξάλλου οι περισσότερες από τις υπηρεσίες που συζητάμε παρέχουν έναν αρχικό, δωρεάν χώρο, που για τους μέσους χρήστες είναι συνήθως αρκετός. Πέρα από το χρηματικό κόστος, κατά νου έχουμε το αντίτιμο της αμφισβητήσιμης ιδιωτικότητας των αρχείων που ανεβαίνουν στις υπηρεσίες cloud storage. Παρά τις σχετικές διαβεβαιώσεις των παρόχων περί του αντιθέτου, γεγονότα όπως διαρροές κωδικών λόγω προβλημάτων ασφαλείας των υπηρεσιών, έλεγχος των αρχείων από τους παρόχους για περιεχόμενο που παραβιάζει όρους χρήσης, αιτήσεις των Αρχών για πρόσβαση σε λογαριασμούς κ.ά., δεν αφήνουν τους χρήστες να κοιμούνται ήσυχοι — όχι όσον αφορά ζητήματα ιδιωτικότητας.

Ας μη γελιόμαστε, όμως. Για οτιδήποτε ανεβαίνει στο web, ίσως θα πρέπει να ξεχάσουμε την έννοια τις ιδιωτικότητας. Κατά περιπτώσεις, η συνειδητοποίηση αυτή είναι πιθανό να μας αποτρέπει από τη χρήση ορισμένων εξαιρετικά χρήσιμων υπηρεσιών. Εν προκειμένω, αξίζει να απαρνηθούμε την ευχρηστία του συγχρονισμού αρχείων μεταξύ διαφορετικών συσκευών, μέσω Διαδικτύου; Για ορισμένους, ναι, αξίζει, για εμάς όμως σίγουρα όχι. Εξάλλου δεν είμαστε εδώ για να καταστροφολογούμε, αλλά για να βρίσκουμε δημιουργικές κι όσο το δυνατόν βέλτιστες λύσεις. Α, και φυσικά για να το διασκεδάζουμε :)

BitTorrent Sync

Πρόσφατα, πολλές συζητήσεις γίνονται γύρω από το BitTorrent Sync και μόνον αν διαμένετε στο Ανατολικό Τιμόρ (ενδεχομένως και με πρόσβαση στο κυβερνητικό Internet), δεν θα έχει πέσει στην αντίληψή σας. Το BitTorrent Sync κυκλοφορεί



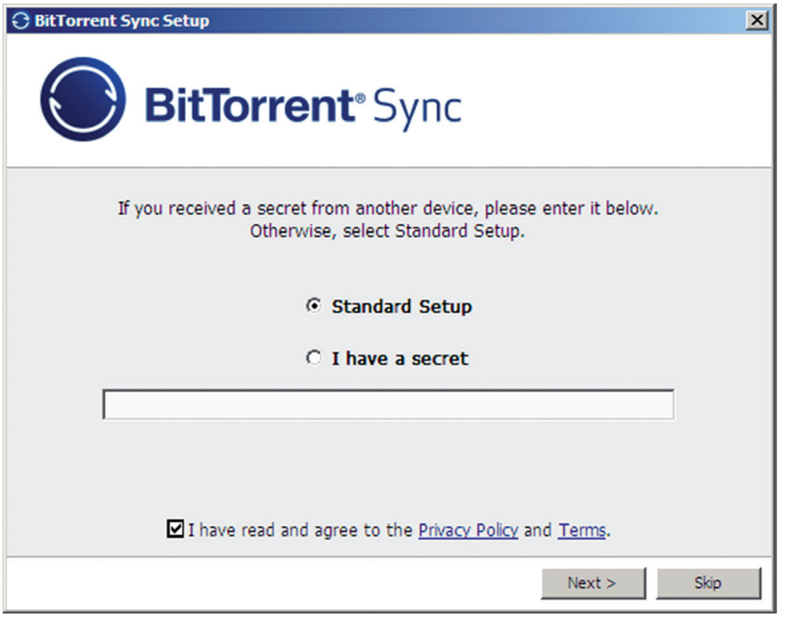
Η προσθήκη ενός φακέλου για συγχρονισμό στο BitTorrent Sync, εκτός των άλλων δημιουργεί έναν μακροσκελή μυστικό κωδικό. Με βάση αυτόν επιτυγχάνεται ο συγχρονισμός με αντίστοιχους φακέλους, σε άλλες συσκευές/υπολογιστές.

από την εταιρεία που βρίσκεται πίσω από το πρωτόκολο του BitTorrent, τον διάσημο torrent client ονόματι BitTorrent, καθώς και τον ακόμα πιο διάσημο client ονόματι μTorrent. Το BitTorrent Sync είναι μια κλειστού κώδικα εφαρμογή, η οποία χρησιμοποιώντας peer-to-peer τεχνικές σε συνδυασμό με υψηλή κρυπτογράφηση επιτρέπει τον ασφαλή συγχρονισμό αρχείων μεταξύ διαφορετικών υπολογιστών/συσκευών. Σίγουρα ο "κλειστός" χαρακτήρας του κώδικα είναι κάτι που μπορεί να προβληματίσει, όμως αντάξια, ανοικτού κώδικα, εναλλακτική λύση ακόμα δεν υπάρχει (ή για να ακριβολογούμε, δεν υπάρχει κάποια που να 'χει φτάσει στο υψηλό επίπεδο αξιοπιστίας του BitTorrent Sync).

Να σημειώσουμε εδώ ότι περιορισμοί ως προς το πλήθος των συσκευών που συγχρονίζονται ή ως προς τον όγκο των διακινούμενων δεδομένων, στο BitTorrent Sync δεν υπάρχουν. Αυτό βεβαίως στη θεωρία, μιας και στην πράξη πρέπει να υπάρχει επαρκής χώρος στα αποθηκευτικά μέσα των εμπλεκομένων συσκευών. Αλλά αυτή η παρατήρηση ήταν μάλλον περιττή ;)

Το μόνο "κακό", αν θέλετε, με το BitTorrent Sync, είναι πως για να συγχρονίζονται τα αρχεία δύο απομακρυσμένων μηχανημάτων πρέπει αμφότερα να βρίσκονται σε λειτουργία ταυτόχρονα. Αυτό δεν βολεύει πάντα και είναι κάτι στο οποίο σίγουρα υπερτερούν οι υπηρεσίες cloud storage, όπως το Dropbox. Μια λύση είναι να στήσουμε το BitTorrent Sync *και* σε κάποιο VPS, αλλά δεν είναι η μόνη. Συνεχίστε το διάβασμα...

> *Σημείωση: Για τη συνέχεια θεωρούμε ότι ο αναγνώστης είναι εξοικειωμένος με τον τρόπο χρήσης του BitTorrent Sync, π.χ., επειδή δουλεύει με τον σχετικό client για Windows ή OS X.*



Κατά την πρώτη εκκίνηση του BitTorrent Sync client, ο χρήστης μπορεί να διαλέξει έναν τοπικό φάκελο για συγχρονισμό ή, αν χρησιμοποιεί ήδη το BitTorrent Sync σε άλλη συσκευή/υπολογιστή και θέλει να συγχρονίσει υπάρχοντα φάκελο με κάποιον στον υπολογιστή που τώρα χρησιμοποιεί, θα πρέπει να εισαγάγει τον αντίστοιχο μυστικό κωδικό.

Η δημιουργία ενός BitTorrent Sync box

Και με μια περιστροφή της καρέκλας, το μάτι πέφτει στο συνήθη ύποπτο: Είναι το Raspberry Pi, το οποίο αν και καταταλαιπωρημένο συνεχίζει να κάθεται ήσυχο και να επιτελεί υπομονετικά το έργο του, εκεί, στην άκρη του δωματίου. Καταλαβαίνετε που πάει το μυαλό μας, έτσι δεν είναι; Σαφώς! Το Raspberry Pi καταναλώνει ελάχιστη ενέργεια και θα αποτελέσει τον απόλυτο σύμμαχό μας, στο στήσιμο ενός ενδιάμεσου, always-on μέσου συγχρονισμού. Για μια ακόμη φορά, σηκώνουμε τα μανίκια κι ετοιμαζόμαστε να περάσουμε ένα ευχάριστο, αν μη τι άλλο, απόγευμα!

Για αρχή θα πρέπει να συνδεθούμε, π.χ., μέσω SSH, στην κονσόλα του Raspberry Pi και να κατεβάσουμε τον κατάλληλο BitTorrent Sync client για επεξεργαστές αρχιτεκτονικής ARM, όπως αυτός που έχει το SoC του RasPi. (Η διανομή Linux που τρέχει το δικό μας Raspberry Pi είναι το Raspbian.) Χρησιμοποιώντας λοιπόν το εργαλείο wget, δίνουμε:

```
$ cd
$ wget http://download-lb.utorrent.com/endpoint/btsync/os/linux-arm/track/stable
$ mv stable btsync_arm.tar.gz
```

Την εντολή mv τη χρησιμοποιούμε ώστε να μετονομάσουμε το αρχείο ονόματι stable που κατέβηκε, στο "φυσιολογικό" όνομα του συμπιεσμένου archive. Ακολούθως αποσυμπιέζουμε με

```
$ tar -zxvf btsync_arm.tar.gz
```

Το εκτελέσιμο αρχείο btsync που προκύπτει μεταφέρουμε στον κατάλογο ~/.sync:

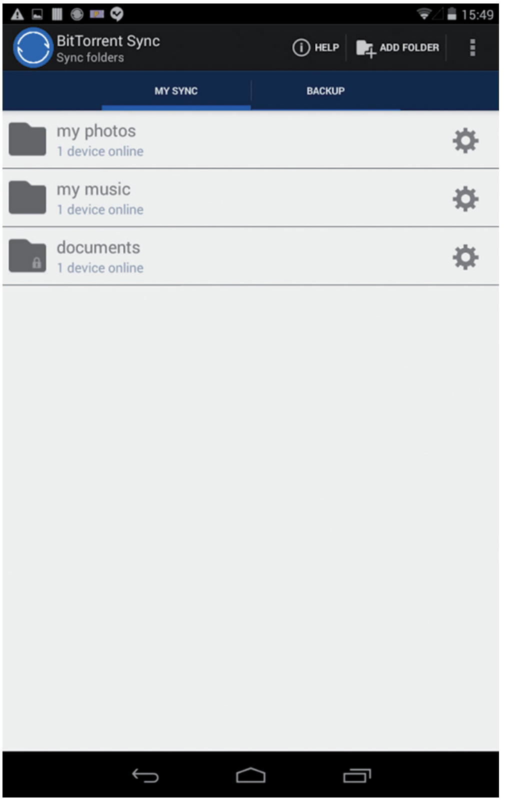
```
$ mkdir ~/.sync
$ mv btsync ~/.sync
```

Τώρα βέβαια μένει να καταφέρουμε να πείσουμε το BitTorrent Sync να εκκινεί μαζί με το σύστημα. Προς τούτο, με δικαιώματα root δημιουργούμε το κατάλληλο script στον κατάλογο /etc/init.d

```
$ sudo nano /etc/init.d/btsync
```

και του εισάγουμε τις ακόλουθες γραμμές, οι οποίες αποτελούν το script εκκίνησης και τερματισμού του btsync:

```
#! /bin/sh
case "$1" in
start)
    /home/pi/.sync/btsync
    ;;
stop)
    killall btsync
    ;;
*)
    echo "Usage: /etc/init.d/btsync {start|stop}"
```



Το BitTorrent Sync, πέραν από desktop και console έχει και clients για φορητές συσκευές που βασίζονται στα δημοφιλή λειτουργικά Android και iOS. Εδώ βλέπουμε το περιβάλλον του client σε μια συσκευή Android.

```
    exit 1
    ;;
esac
exit 0
```

Κάνουμε το script εκτελέσιμο και το καταχωρούμε ως υπηρεσία εκκίνησης του συστήματος:

```
$ sudo chmod +x /etc/init.d/btsync
$ sudo update-rc.d btsync defaults
```

Μπορούμε τώρα να ξεκινήσουμε το BitTorrent Sync με μια επανεκκίνηση στο Raspberry — ή να το ξεκινήσουμε απευθείας δίνοντας

```
$ sudo service btsync start
```

Και τώρα που εκκινήσαμε το BitTorrent Sync μπορούμε να μπούμε στο web panel διαχείρισής του, το οποίο είναι προσβάσιμο μέσω του αγαπημένου μας web browser. Όπως φαντάζεστε, ο client του BitTorrent Sync σηκώνει έναν δικό του web server, ο οποίος ακούει στη θύρα 8888/TCP.

Από κάποιον άλλον υπολογιστή, λοιπόν, ξεκινάμε τον αγαπημένο μας web browser και στη μπάρα διευθύνσεων πληκτρολογούμε το IP του Raspberry, ακολουθούμενη από άνω-κάτω τελεία και τη θύρα επικοινωνίας (8888) με το όμορφο web interface του BitTorrent Sync (το οποίο, ας σημειωθεί, αξιοποιεί για την εμφάνισή του το Bootstrap framework). Αν, π.χ., το RaspPi έχει ως IP το 192.168.1.10, στη μπάρα του browser δίνουμε

```
http://192.168.1.10:8888
```

Από το WebUI του BitTorrent Sync μπορούμε να προσθέσουμε καταλόγους προς συγχρονισμό. Κάποιες ρυθμίσεις, καθώς και τα της προστασίας του web panel με κωδικό, μπορούμε να τις κάνουμε από αυτό το περιβάλλον. Ειδικά για τον κωδικό πρόσβασης στο WebUI, οι προγραμματιστές του BitTorrent Sync έχουν μεριμνήσει έτσι ώστε να μας ζητηθεί να τον προσθέσουμε από την πρώτη κιόλας επίσκεψη. Επιπλέον ρυθμίσεις μπορούμε να κάνουμε αξιοποιώντας ένα σχετικό αρχείο της υπηρεσίας — αφού όμως πρώτα το δημιουργήσουμε. Για να κάνουμε κάτι τέτοιο, σταματάμε πρώτα την υπηρεσία με

```
$ sudo /etc/init.d/btsync stop
```

και μετά δίνουμε

```
$ ~/.sync/btsync --dump-sample-config > ~/.sync/btsync.conf
```

Το παράδειγμα του αρχείου ρυθμίσεων που μας δίνει το btsync χρησιμοποιεί την, ιδιαιτέρως αγαπητή στο γράφοντα, μορφή JSON. Ιδού τα περιεχόμενά του:

```
{
  "device_name": "My Sync Device",
  "listening_port" : 0,     // 0 - randomize port
/* storage_path dir contains auxilliary app files
   if no storage_path field: .sync dir created in the directory
   where binary is located.
   otherwise user-defined directory will be used
*/
  "storage_path" : "/home/user/.sync",
// uncomment next line if you want to set location of pid file
// "pid_file" : "/var/run/btsync/btsync.pid",
  "check_for_updates" : true,
  "use_upnp" : true,         // use UPnP for port mapping
/* limits in kB/s
   0 - no limit
*/
  "download_limit" : 0,
  "upload_limit" : 0,
/* remove "listen" field to disable WebUI
   remove "login" and "password" fields to disable credentials check
*/
  "webui" :
  {
    "listen" : "0.0.0.0:8888",
    "login" : "admin",
    "password" : "password"
  }
/* !!! if you set shared folders in config file WebUI will be DISABLED !!!
   shared directories specified in config file
   override the folders previously added from WebUI.
*/
/*
```

```
  ,
  "shared_folders" :
  [
    {
//  use --generate-secret in command line to create new secret
      "secret" : "MY_SECRET_1",    // * required field
      "dir" : "/home/user/bittorrent/sync_test", // * required field
//  use relay server when direct connection fails
      "use_relay_server" : true,
      "use_tracker" : true,
      "use_dht" : false,
      "search_lan" : true,
//  enable SyncArchive to store files deleted on remote devices
      "use_sync_trash" : true,
//  restore modified files to original version, ONLY for Read-Only folders
//    "overwrite_changes" : false,
//  specify hosts to attempt connection without additional search
      "known_hosts" :
      [
        "192.168.1.2:44444"
      ]
    }
  ]
*/
}
```

Αφού πραγματοποιήσουμε τις επιθυμητές αλλαγές οφείλουμε να ενημερώσουμε και το script εκκίνησης /etc/init.d/btsync, τροποποιώντας τη γραμμή της περίπτωσης "start" ως εξής:

```
/home/pi/.sync/btsync --config btsync.conf
```

Θα εκκινήσουμε ξανά[1], φυσικά, την υπηρεσία:

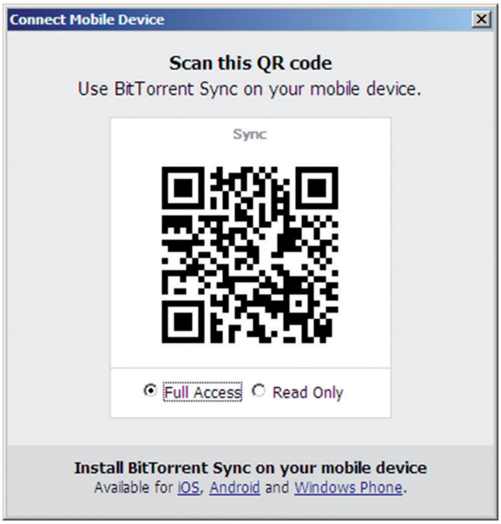
```
$ sudo /etc/init.d/btsync start
```

BitTorrent Sync και ownCloud: Μια σχέση λατρείας!

Μέχρι στιγμής έχουμε καταφέρει να στήσουμε έναν πλήρως λειτουργικό BitTorrent Sync client στο Raspberry Pi μας, μετατρέποντάς το έτσι σε μια ενδιάμεση, always-

1. Οι εθισμένοι στα Windows, μπορούν να προβούν και σε μια επανεκκίνηση του συστήματος :P

on συσκευή συγχρονισμού των αρχείων μας. Φυσικά δεν θα μείνουμε εδώ, καθώς πρόθεσή μας είναι η δημιουργία μιας λύσης που τίποτε δεν θα έχει να ζηλέψει από τις αντίστοιχες υπηρεσίες online storage, όπως, π.χ., είναι το δημοφιλέστατο Dropbox. Στην προκειμένη περίπτωση, αυτό που μας λείπει είναι ένα web panel για την πρόσβαση στα αποθηκευμένα αρχεία. Έτσι, θα έχουμε τη δυνατότητα πρόσβασης σ' αυτά ακόμη κι όταν δεν χρησιμοποιούμε κάποια από τις δικές μας συσκευές ή υπολογιστές: Το μόνο που θα χρειαζόμαστε είναι έναν web browser (και φυσικά πρόσβαση στο Intetnet). Για μια ακόμη φορά, θα στραφούμε στο ownCloud[2].



Ο συγχρονισμός ενός φακέλου με μια φορητή συσκευή είναι κάτι παραπάνω από εύκολος, μιας και οι προγραμματιστές του BitTorrent Sync έχουν απλοποιήσει τη διαδικασία εισαγωγής του μακροσκελούς, μυστικού κωδικού με τη χρήση QR codes.

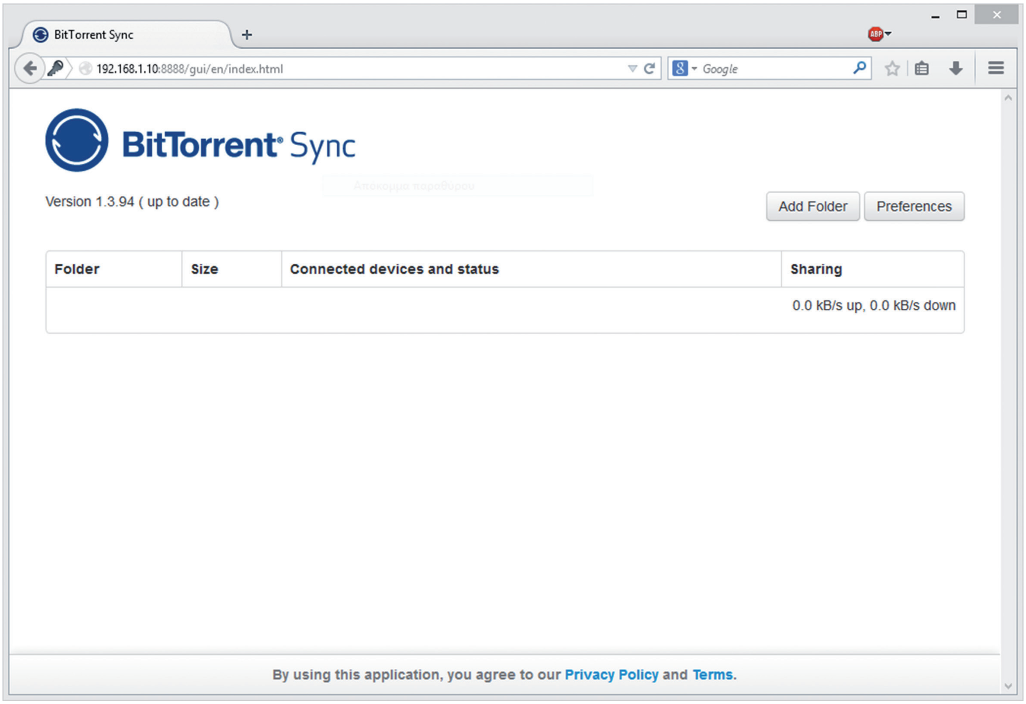
Για την εγκατάσταση του ownCloud στο RaspPi θα χρησιμοποιήσουμε την αυτοματοποιημένη λύση του ownCloudPie, το οποίο είναι ένα shell script που κατεβάζει κι εγκαθιστά για εμάς την τελευταία έκδοση του ownCloud *μαζί* με τον Apache ή τον NGiNX. Φυσικά, όσοι έχουν εγκατεστημένο ήδη κάποιον web server ή είναι εξοικειωμένοι με τη χρήση του Raspbian και θέλουν να έχουν τον πλήρη έλεγχο (ή τέλος πάντων να περάσουν ένα ακόμη πιο ευχάριστο απόγευμα), ίσως προτιμήσουν να κάνουν "χειροκίνητα" την εγκατάσταση του ownCloud, αντί να καταφύγουν στο αυτοματοποιημένο αυτό script.

Όσοι πάντως επιλέξουν την οδό του ownCloudPie, οφείλουν να εγκαταστήσουν το git (αν δεν το έχουν ήδη), να κατεβάσουν μέσω αυτού το σχετικό script, να το εκτελέσουν και ν' απαντήσουν σε μερικές ερωτήσεις που θα τους απευθυνθούν, σχετικά με την εγκατάσταση. Δείτε:

```
$ sudo apt-get install -y git dialog
$ cd
$ git clone git://github.com/petrockblog/OwncloudPie.git
$ cd OwncloudPie
$ chmod +x owncloudpie_setup.sh
$ sudo ./owncloudpie_setup.sh
```

Από τη στιγμή που το ownCloud, με τον έναν ή τον άλλο τρόπο, βρεθεί εγκατεστημένο στον λιλιπούτειο, ακούραστο φίλο μας, σειρά έχει η αρμονική συνεργασία του με το BitTorrent Sync. Και για να επιτευχθεί αυτό θα πρέπει, στον BitTorrent Sync client, να προσθέσουμε ως φάκελο συγχρονισμού τον data folder που έχουμε ορίσει στο ownCloud. Προσοχή: Αν το data folder του ownCloud δεν έχει τα κατάλληλα δικαιώματα εγγραφής για τον user που χρησιμοποιεί το BitTorrent Sync, με τη βο-

2. Για περισσότερες λεπτομέρειες σχετικά με το ownCloud μπορείτε να διαβάσετε το άρθρο του φίλτατου GiannouG, "Το Dropbox του φτωχού", στο τεύχος 019 του deltaHacker.



Το λιτό αλλά όμορφο web panel διαχείρισης του BitTorrent Sync, μετά την εγκατάστασή του στο Raspberry Pi μας!

ήθεια του chmod θα πρέπει να τα καθορίσουμε. Τέλος, οφείλουμε να κάνουμε τις απαραίτητες ρυθμίσεις port forwarding στον router μας, έτσι ώστε το ownCloud να είναι προσβάσιμο κι από το Internet.

Ενίσχυση ασφάλειας

Εφόσον η ασφάλεια και η ιδιωτικότητα είναι τα κύρια μελήματά μας, θα είναι φρόνιμο να διατηρούμε πάντα πλήρως ενημερωμένο το λογισμικό που σερβίρει το ownCloud. Αν βέβαια θέλουμε να κρατήσουμε απομονωμένα τα αρχεία μας από τον έξω κόσμο, δηλαδή να είναι προσβάσιμα μόνον για τα μηχανήματα του τοπικού δικτύου, τότε σίγουρα δεν θα καταφύγουμε στο port fowrading. Εναλλακτικά, έχουμε τη δυνατότητα να συνδεόμαστε από οπουδήποτε και με απόλυτη ασφάλεια στο τοπικό δίκτυο — βεβαίως και να κάνουμε χρήση όλων των υπηρεσιών που προσφέρονται εντός αυτού: Αρκεί να στήσουμε το OpenVPN! Αλλά περισσότερα επ' αυτού μπορείτε να διαβάσετε στο σχετικό άρθρο που δημοσιεύεται στο `http://deltahacker.gr/?p=5058`.

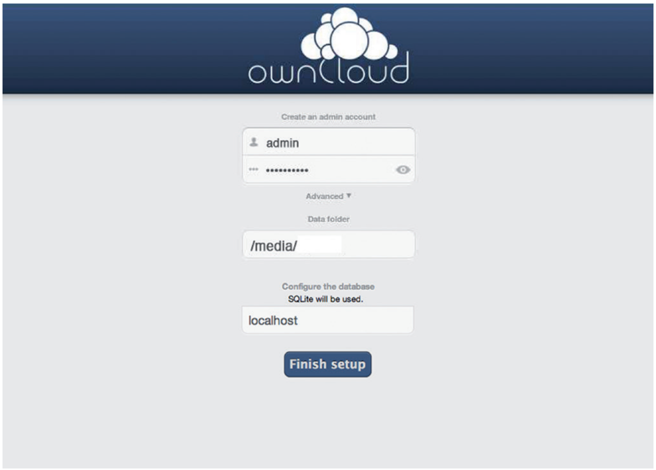
Σημείωση: Αν επιθυμούμε να περιορίσουμε το συγχρονισμό μέσω BitTorrent Sync εντός LAN και μόνο, τότε από τις ρυθμίσεις του Bittornent Sync client θα πρέπει να απενεργοποιήσουμε τη χρήση relay server, tracker server καθώς και την αναζήτηση DHT (αφήνοντας όμως ενεργή την αναζήτηση στο τοπικό δίκτυο).

The job is done

Αυτό ήταν! Ο στόχος μας επιτεύχθηκε άμεσα και γρήγορα, με τον συνήθη ύποπτο[3] να βγάζει πάλι το φίδι απ' την τρύπα. Φυσικά, οι βελτιώσεις που επιδέχεται η υλοποίησή μας είναι πολλές.

Σκεφτόμαστε, π.χ., το θέμα της ταχύτητας συγχρονισμού, η οποία για boxes/VMs εκτός τοπικού δικτύου θα περιορίζεται στην ταχύτητα upload της ιντερνετικής σύνδεσης — εκτός κι αν είστε από τους τυχερούς που μπορούν κι απολαμβάνουν καλωδιακό Internet. Αν, πάντως, παράλληλα με το RaspPi, στήσετε το BitTorrent Sync και σε κάποιο VPS (που βεβαίως διαθέτει ικανό αποθηκευτικό χώρο), τότε αφενός θα απολαμβάνετε μέγιστες ταχύτητες συγχρονισμού εντός LAN, αφετέρου εκμεταλλευόμενοι την επικοινωνία του always-on Raspberry Pi με το VPS θα έχετε ιδιαίτερα αυξημένες ταχύτητες συγχρονισμού ακόμη κι όταν βρίσκεστε εκτός LAN. Κατ' αυτόν τον τρόπο, η λύση σας πραγματικά δεν θα έχει τίποτε να ζηλέψει από υπηρεσίες όπως αυτή του Dropbox. Αν μάλιστα το VPS βρίσκεται σε κάποιον IaaS provider με άριστη φήμη όσον αφορά στην προστασία των δεδομένων των πελατών του[4], τότε εξασφαλίζουμε στο μέγιστο βαθμό τα δεδομένα μας. Επιπρόσθετα, αν ο IaaS provider προσφέρει ένα κάποιο ευέλικτο μοντέλο cloud storage, ανά πάσα στιγμή θα έχουμε ακριβώς όσο χώρο χρειαζόμαστε και —κατά περιπτώσεις— με εξαιρετικά συμφέρον κόστος. Περισσότερα όμως για όλα αυτά θα συζητήσουμε σε επόμενα άρθρα ή/και επεισόδια του deltaCast (http://deltahacker.gr/category/deltacast).

Το μόνο που μας ενοχλεί λίγο με το BitTorrent Sync είναι ότι δεν αποτελεί Ανοικτό Λογισμικό. Αυτό δεν είναι κατ' ανάγκη κακό και μάλιστα έως σήμερα δεν υπάρχει τίποτε που να υποδεικνύει ότι το BitTorrent Sync έχει κάποιο σοβαρό bug ή περιλαμβάνει, π.χ., κάποιο backdoor. Θα μας άρεσε μολαταύτα να έχουμε διαθέσιμο τον κώδικά του για μελέτη — γιατί όχι και για βελτίωση. Ευτυχώς, δεν είμαστε οι μόνοι με παρόμοιες ανησυχίες. Έτσι, σιγά σιγά έχουν αρχίσει ν' αναπτύσσονται αντίστοιχες λύσεις Ανοικτού Λογισμικού. Μία απ' αυτές που ξεχωρίσαμε είναι το Syncthing, το οποίο όμως έχει αρκετό δρόμο ακόμα να διανύσει. Είμαστε πάντως βέβαιοι ότι σίγουρα θα μας απασχολήσει σε μελλοντικό άρθρο — ή τουλάχιστον αυτό επιθυμούμε :)



Ο φάκελος που επιλέγουμε να "αντλεί" αρχεία το ownCloud, είναι ο ίδιος με εκείνον που προσθέσαμε προς συγχρονισμό με το BitTorrent Sync.

3. Ο οποίος δεν είναι άλλος από το Raspberry Pi.
4. Σημαντική εδώ είναι και η νομοθεσία της χώρας στην οποία βρίσκεται το datacenter του provider!

Skill: Intermediate
Tags: symmetric key, public key, Diffie-Hellman, pseudorandomness

ΤΥΧΑΙΟΙ ΑΡΙΘΜΟΙ

μέρος 1ο

by George of the Jungle

Η τύχη στην Κρυπτογραφία

Θα μπορούσαμε να ισχυριστούμε ότι οι σύγχρονες μέθοδοι κρυπτογράφησης προσφέρουν εξαιρετικά μεγάλη ασφάλεια [*]. Με την επεξεργαστική ισχύ που διαθέτουμε σήμερα, η αποκρυπτογράφηση ενός μηνύματος χωρίς τα αντίστοιχα κλειδιά, είναι πρακτικά αδύνατη. Ε, λοιπόν, οι μέθοδοι κρυπτογράφησης στις οποίες εμπιστευόμαστε τα μυστικά μας, στηρίζονται σε μεγάλο βαθμό στην παραγωγή τυχαίων αριθμών. Κι όσο πιο τυχαίοι είναι αυτοί οι αριθμοί, τόσο πιο ήσυχοι μπορούμε να είμαστε για την ασφάλεια των μηνυμάτων μας.

[*] Τα bugs που κατά καιρούς ανακαλύπτονται στα σχετικά προγράμματα δεν αφορούν τις ίδιες τις μεθόδους της κρυπτογράφησης αλλά τις υλοποιήσεις τους.

Η εισαγωγή του άρθρου ενδέχεται να σας δημιούργησε ήδη μια απορία: Τι εννοούμε όταν κάνουμε λόγο για περισσότερο ή λιγότερο τυχαίους αριθμούς; Είναι δυνατό ένα σύνολο αριθμών να είναι "λίγο τυχαίο"; Η κοινή λογική υπαγορεύει ότι ένα σύνολο αριθμών μπορεί είτε να είναι, είτε να μην είναι τυχαίο. Ενδιάμεσες καταστάσεις δεν υπάρχουν. Και μη νομίζετε ότι αποφασίσαμε να τα βάλουμε με την κοινή λογική. Απλά, συνηθίζουμε να επιμένουμε σε λεπτομέρειες που διαφεύγουν από όσους επικαλούνται με το παραμικρό την κοινή λογική. Έτσι, θα απαντήσουμε στα αρχικά —κι ομολογουμένως δικαιολογημένα— ερωτήματα με ένα άλλο ερώτημα: Πότε μπορούμε να πούμε ότι ένα σύνολο αριθμών είναι τυχαίο; Αν δεν ορίσουμε προσεκτικά την έννοια του τυχαίου, είναι αδύνατο να απαντήσουμε στο αν η τυχαιότητα μπορεί να αξιολογηθεί ή, τέλος πάντων, να μετρηθεί. Μήπως σας μπερδέψαμε περισσότερο τώρα; Δεν πειράζει. Ελπίζουμε μόνο να κινήσαμε και την περιέργειά σας.

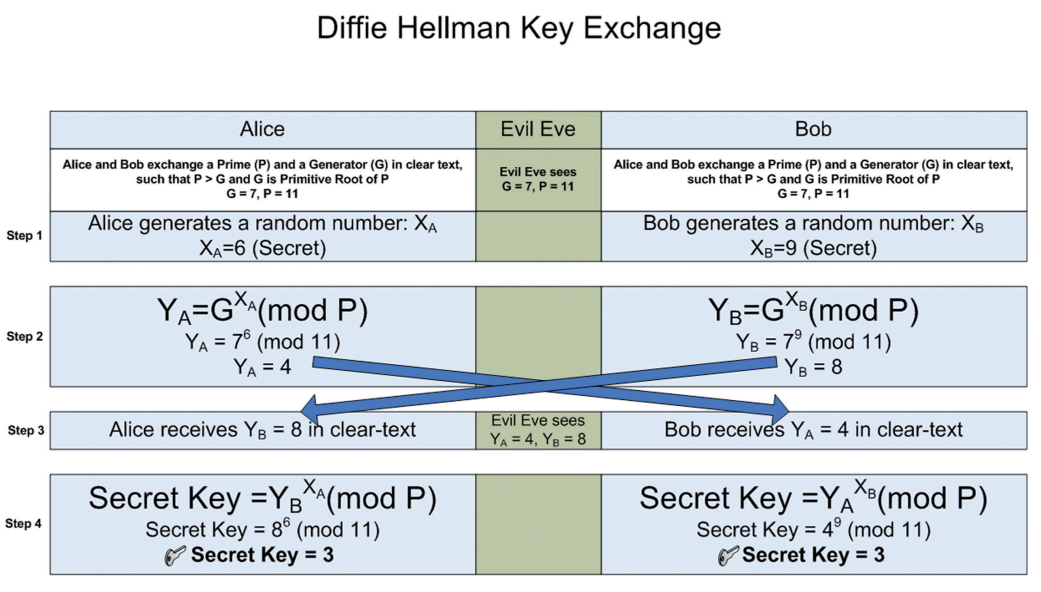
Περί κρυπτογραφίας

Η έννοια της κρυπτογραφίας είναι λίγο-πολύ γνωστή σε όλους. Με απλά λόγια, πρόκειται για τη μελέτη και την ανάπτυξη διαφόρων τεχνικών που πετυχαίνουν την κρυπτογράφηση και την αποκρυπτογράφηση. Δηλαδή το μετασχηματισμό ενός μηνύματος σε μια ακατάληπτη μορφή (cipher text), που δεν θυμίζει σε τίποτα το αρχικό μήνυμα (clear text) κι αντίστροφα. Για την επίτευξη της αποκρυπτογράφησης δεν αρκεί να γνωρίζουμε τη μέθοδο κρυπτογράφησης. Αν ίσχυε κάτι τέτοιο, η προστασία των κρυπτογραφημένων μηνυμάτων θα ήταν απλά ανύπαρκτη. Για την αποκρυπτογράφηση απαιτείται και η γνώση ενός μυστικού συνόλου δεδομένων, που αποτελούν το λεγόμενο κλειδί. Οι ορισμοί που αναφέραμε, αν και είναι σωστοί, είναι μάλλον απλοϊκοί και περιγράφουν κυρίως τα πρώτα στάδια ανάπτυξης της κρυπτογραφίας. Τεχνικές για τη "συσκότιση" ενός μηνύματος και την επαναφορά του στην αρχική μορφή, υπάρχουν από την αρχαιότητα. Η σύγχρονη κρυπτογραφία καταπιάνεται και με άλλα ζητήματα, όπως είναι η ταυτοποίηση του αποστολέα ενός μηνύματος (authentication), η εξασφάλιση ότι το μήνυμα δεν έχει αλλοιωθεί καθόλου (data integrity) κ.ο.κ. Από τον Πρώτο Παγκόσμιο Πόλεμο και μετά, η κρυπτογραφία έχει συνδεθεί στενά με τους υπολογιστές και τα μαθηματικά. Οι σημερινές μέθοδοι κρυπτογράφησης στηρίζονται σε περίπλοκες μαθηματικές διαδικασίες, οι οποίες αν και θεωρητικά είναι πράγματι δυνατό ν' αντιστραφούν κι έτσι η κρυπτογράφηση να "σπάσει", για κάθε πρακτική εφαρμογή δεχόμαστε ότι είναι "αρκετά ασφαλείς". Αυτό συμβαίνει γιατί η επίλυση των σχετικών μαθηματικών προβλημάτων είναι εξαιρετικά δύσκολη, απαιτεί τεράστια υπολογιστική ισχύ και τόσο μεγάλο χρόνο, που η αποκρυπτογράφηση καθίσταται πρακτικά αδύνατη. Καλά όλα αυτά, θα πείτε, αλλά που κολλάνε οι τυχαίοι αριθμοί;

Μεταφορά κλειδιού

Όπως αναφέραμε, για την αποκρυπτογράφηση ενός μηνύματος πρέπει να γνωρίζουμε το κλειδί με το οποίο έγινε η κρυπτογράφηση. Αυτό ισχύει για όλες τις παλιές τεχνικές κρυπτογράφησης, που χαρακτηρίζονται ως τεχνικές "συμμετρικού κλειδιού" (Symmetric Key Cryptography), ακριβώς επειδή η κρυπτογράφηση και η αποκρυπτογράφηση πραγματοποιούνται με το ίδιο κλειδί. Ας έρθουμε τώρα στην εποχή μας, με όλη αυτή την αφθονία υπολογιστικής ισχύος. Οι τεχνικές "συμμετρικού κλειδιού" προσφέρουν μεγάλη ασφάλεια, αλλά παρουσιάζουν ένα πρόβλημα: Πώς θα αποκτήσουν οι δύο "συνομιλητές" το απαραίτητο κλειδί; Πώς μπορούμε να εξασφαλίσουμε ότι δεν θα κλαπεί κατά τη μεταφορά, εφόσον το Internet είναι εν γένει

επισφαλές μέσο; Προφανώς, το να βρεθούν οι δύο ενδιαφερόμενοι από κοντά και να ανταλλάξουν το κλειδί χέρι με χέρι, δεν αποτελεί καθόλου πρακτική λύση. Εξάλλου, δεν αποκλείεται να βρίσκονται σε διαφορετικές πόλεις, χώρες ή και ηπείρους. Αυτό το πρόβλημα λύθηκε από τους Whitfield Diffie και Martin Hellman, με έναν αλγόριθμο που δημοσίευσαν το 1976 κι ο οποίος πήρε το όνομά τους (Diffie-Hellman). Ο συγκεκριμένος αλγόριθμος επιτρέπει σε δύο άτομα να ανταλλάξουν δύο αριθμούς μέσα από ένα μη ασφαλές δίκτυο επικοινωνίας, και με αυτούς να δημιουργήσουν ένα κοινό κλειδί που θα είναι γνωστό *αποκλειστικά* σε αυτούς. Στη συνέχεια, με το συγκεκριμένο κλειδί θα μπορούν να κρυπτογραφούν και να αποκρυπτογραφούν τα μηνύματά τους, χρησιμοποιώντας μια τεχνική "συμμετρικού κλειδιού". Κάπου εδώ είναι λογικό να προκύψει το εξής ερώτημα: Αν κάποιος επιτήδειος κλέψει τους αριθμούς που ανταλλάσσουν οι συνομιλητές, δεν θα μπορέσει κι αυτός να συνθέσει το κλειδί; Η απάντηση είναι αρνητική — τουλάχιστον στην πράξη ;) Αλλά ας δούμε πώς δουλεύει ο αλγόριθμος Diffie-Hellman και θα καταλάβετε το γιατί.



Το πρωτόκολλο των Diffie-Hellman παρουσιάστηκε το 1976 από τους Whitfield Diffie και Martin Hellman. Επιτρέπει σε δυο χρήστες, χωρίς προηγούμενη επικοινωνία, να συνθέσουν ένα κοινό κλειδί, ανταλλάσσοντας ορισμένα άλλα μεγέθη μέσω ενός μη ασφαλούς μέσου επικοινωνίας.

Αν μπορείς, βρες το!

Ας υποθέσουμε ότι ο Χρήστος θέλει να ανταλλάξει κρυπτογραφημένα μηνύματα με τον Γιώργο, με κάποια τεχνική "συμμετρικού κλειδιού". Σύμφωνα με τον αλγόριθμο Diffie-Hellman, πρέπει να γίνουν τα ακόλουθα:

- Ο Χρήστος υπολογίζει ένα *τυχαίο* μυστικό κλειδί (x), το οποίο δεν πρόκειται να αποκαλύψει ποτέ και σε κανέναν.
- Ο Γιώργος υπολογίζει κι αυτός ένα *τυχαίο* μυστικό κλειδί (y), το οποίο επίσης θα διατηρήσει κρυφό από τους πάντες, εις τον αιώνα τον άπαντα.

- Ο Χρήστος υπολογίζει δύο ακέραιους αριθμούς, τους g και p, οι οποίοι έχουν συγκεκριμένες μαθηματικές ιδιότητες. Εμείς δεν θα μπλέξουμε τώρα με τη Θεωρία Αριθμών και δεν θα ασχοληθούμε περισσότερο με τη "φύση" των g και p. Στη συνέχεια, ο Χρήστος στέλνει στο Γιώργο τα μεγέθη: g, p και X = (g^x) mod p.
- Ο Γιώργος λαμβάνει τα g και p, υπολογίζει το Y = (g^y) mod p και το στέλνει στο Χρήστο.

Εδώ πρέπει να σημειώσουμε το εξής: Αν κάποιος επιτήδειος κλέψει τα νούμερα που ανταλλάσουν οι δύο χρήστες, θα έχει στη διάθεσή του τα μεγέθη Y = (g^y) mod p και X = (g^x) mod p, όπως επίσης τα g και p. Ε, λοιπόν, εξαιτίας του μεγάλου μεγέθους που (πρέπει να) έχει ο p, καθώς κι ορισμένων άλλων ιδιοτήτων που έχουν οι g και p, είναι πρακτικά αδύνατο για τον υποκλοπέα να υπολογίσει τους x και y. Επομένως, τα δύο τυχαία κλειδιά που δημιούργησαν ο Χρήστος κι ο Γιώργος παραμένουν ασφαλή. Ο Χρήστος κι ο Γιώργος, όμως, δεν έχουν κάποιο κοινό κλειδί για την κρυπτογράφηση των μηνυμάτων τους. Τουλάχιστον όχι ακόμα. Σε αυτό το στάδιο απομένει ένα βήμα για να λυθεί το πρόβλημα του κοινού μυστικού κλειδιού. Ο Χρήστος θα υπολογίσει το μέγεθος (Y^x) mod p, ενώ ο Γιώργος θα υπολογίσει το μέγεθος (X^y) mod p. Αυτά τα δύο μεγέθη δεν θα μπορούσε να τα υπολογίσει κανένας άλλος, αφού οι αριθμοί x και y παραμένουν μυστικοί. Κι αν μας επιτρέπετε την έκφραση, η μεγάλη μαγκιά έγκειται στο γεγονός ότι αυτά τα δύο μεγέθη ταυτίζονται! Επομένως, ο Χρήστος και ο Γιώργος έχουν καταφέρει να υπολογίσουν ένα κοινό κλειδί, που δεν θα μπορούσε να υπολογίσει κανένας άλλος. Ας δούμε μια σύντομη επίδειξη.

Καταρχάς, σημειώστε ότι ισχύει το εξής:

```
(a*b) mod k = ((a mod k)*(b mod k)) mod k
```

[Γράφοντας "p mod q", εννούμε την πράξη που μας δίνει το υπόλοιπο (modulo) της διαίρεσης του p με το q.] Κατ' επέκταση, ισχύει και το ακόλουθο:

```
(a^n) mod k = ((a mod k)*(a mod k)*...*(a mod k)) mod k
```

Στο δεύτερο σκέλος της παραπάνω ισότητας, το (a mod k) επαναλαμβάνεται n φορές. Έχοντας αυτά κατά νου, θα εξετάσουμε ξανά τα μεγέθη που υπολογίζουν οι δύο χρήστες, στο τελευταίο στάδιο του αλγόριθμου Diffie-Hellman. Όπως είπαμε, ο Χρήστος έχει λάβει από τον Γιώργο το Y = (g^y) mod p και υπολογίζει το (Y^x) mod p, ενώ ο Γιώργος έχει λάβει το X = (g^x) mod p και υπολογίζει το (X^y) mod p. Δώστε βάση στα ακόλουθα:

```
(Y^x) mod p = ((g^y mod p)^x) mod p =
= ((g^y mod p) * (g^y mod p) *...* (g^y mod p)) mod p =
= (g^(y*x)) mod p
```

Ομοίως:

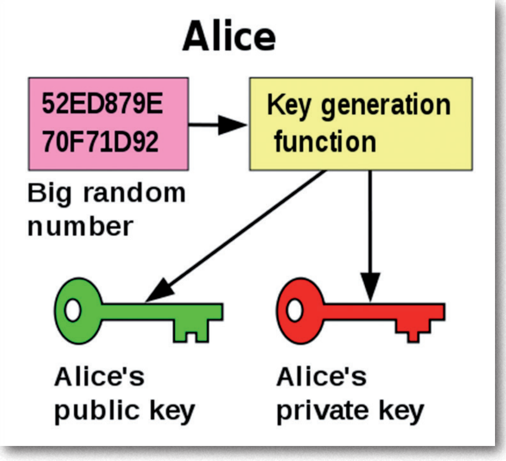
```
(X^y) mod p = ((g^x mod p)^y) mod p =
= ((g^x mod p) * (g^x mod p) *...* (g^x mod p)) mod p =
= (g^(x*y)) mod p
```

Είναι φανερό, λοιπόν, ότι τα μεγέθη που υπολογίζουν ο Χρήστος κι ο Γιώργος ταυτίζονται. Κάπου εδώ πρέπει να τονίσουμε το εξής: Αν οι αριθμοί x και y προέκυπταν από κάποια προβλέψιμη διαδικασία, ο αλγόριθμος θα αχρηστευόταν. Είναι πολύ

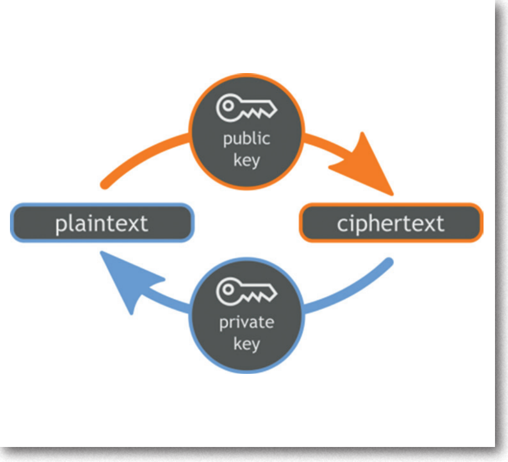
σημαντικό, λοιπόν, οι αριθμοί x και y να είναι εντελώς απρόβλεπτοι ή, τέλος πάντων, "εντελώς τυχαίοι". Αν θέλετε να μάθετε περισσότερα για τον αλγόριθμο Diffie-Hellman μπορείτε να μελετήσετε και το κείμενο "Αυθεντικοποιημένη Εδραίωση Κλειδιού", που θα βρείτε στο `http://bit.ly/authKeyEst`. Επιπρόσθετα, αν σας ενδιαφέρει η ασφάλεια του εν λόγω αλγορίθμου, δείτε κι αυτό: `http://bit.ly/DHsecIssues`.

Μη-συμμετρική κρυπτογραφία

Οι τυχαίοι αριθμοί πρωταγωνιστούν και σε μια άλλη κατηγορία κρυπτογράφησης, η οποία είναι περισσότερο διαδεδομένη. Μιλάμε για την κρυπτογράφηση "δημοσίου κλειδιού" (Public Key Cryptography) ή, αλλιώς, κρυπτογράφηση "μη-συμμετρικού κλειδιού" (Asymmetric Key Cryptography). Η εν λόγω κρυπτογράφηση επινοήθηκε επίσης από τους Whitfield Diffie και Martin Hellman, και περιγράφηκε στην ίδια εργασία του 1976. Εδώ δεν χρησιμοποιείται το ίδιο κλειδί για την κρυπτογράφηση και την αποκρυπτογράφηση, αλλά δύο διαφορετικά. Κάθε χρήστης έχει ένα κλειδί που ονομάζεται ιδιωτικό (Private Key) κι ένα ακόμα που ονομάζεται δημόσιο (Public Key). Όπως αντιλαμβάνεστε, το πρώτο πρέπει να διατηρείται κρυφό, ενώ το δεύτερο μπορεί να διανέμεται ελεύθερα. Αυτά τα δύο κλειδιά δεν είναι άσχετα μεταξύ τους. Το δημόσιο κλειδί παράγεται από το ιδιωτικό αλλά με τέτοιον τρόπο, ώστε η αντίστροφη διαδικασία να είναι πρακτικά αδύνατη. Έτσι, για κάποιον που διαθέτει ένα δημόσιο κλειδί, είναι πρακτικά αδύνατο να υπολογίσει το αντίστοιχο ιδιωτικό. Για την ασφαλή ανταλλαγή μηνυμάτων, ο εκάστοτε αποστολέας πρέπει να κρυπτογραφήσει το μήνυμά του με το δημόσιο κλειδί του παραλήπτη. Κάνοντας κάτι τέτοιο, το μήνυμα θα μπορεί να αποκρυπτογραφηθεί *μόνον* από αυτόν που κατέχει το αντίστοιχο ιδιωτικό κλειδί.



Η δημιουργία του δημοσίου και του ιδιωτικού κλειδιού γίνεται από ειδικές συναρτήσεις, οι οποίες δέχονται σαν είσοδο έναν μεγάλο και τυχαίο αριθμό. Όσο πιο απρόβλεπτος είναι αυτός ο αριθμός, τόσο πιο ασφαλή είναι και τα παραγόμενα κλειδιά.

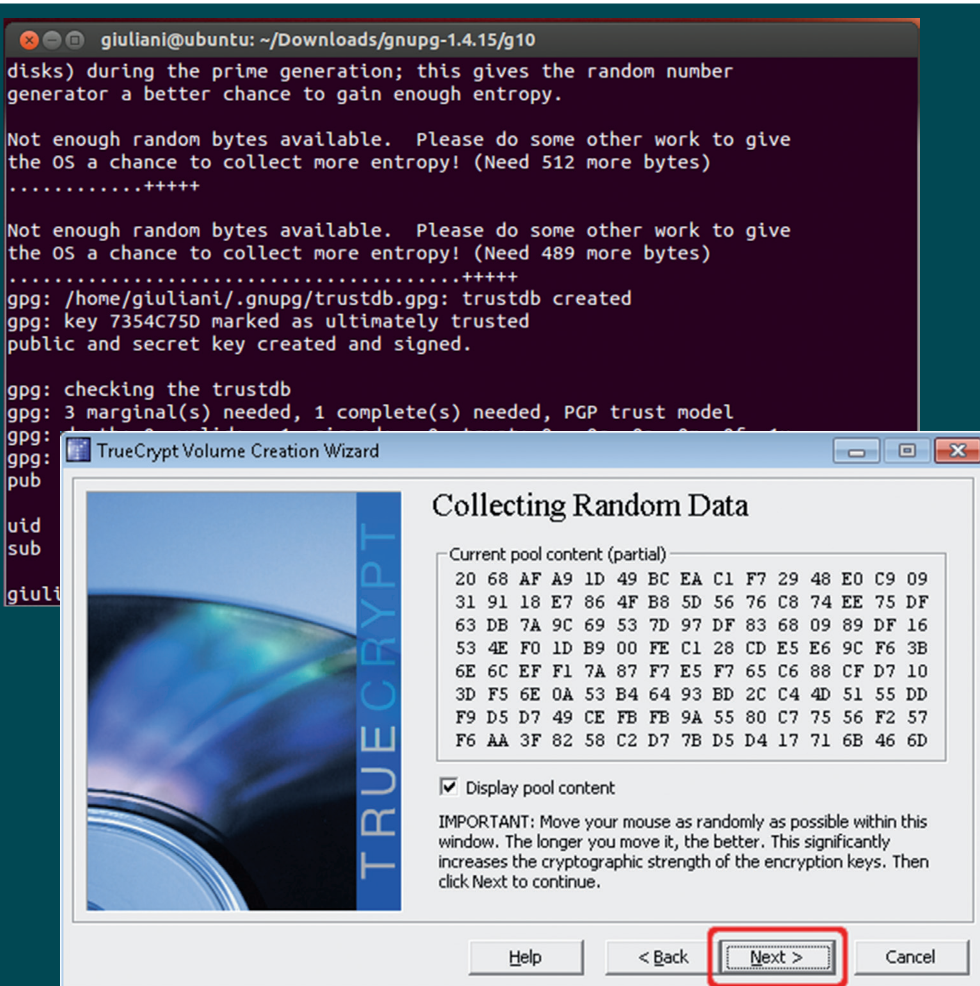


*Στην κρυπτογράφηση δημόσιου-ιδιωτικού κλειδιού, για την αποστολή ενός μηνύματος απαιτείται το δημόσιο κλειδί του παραλήπτη. Το κρυπτογραφημένο μήνυμα θα μπορεί να αποκρυπτογραφηθεί *μόνον* από τον ιδιοκτήτη του αντίστοιχου ιδιωτικού κλειδιού.*

Όπως καταλαβαίνετε, το περισσότερο ενδιαφέρον σε αυτή τη μέθοδο κρυπτογράφησης συγκεντρώνεται στον τρόπο παραγωγής του δημόσιου κλειδιού. Η σχετική διαδικασία είναι, πρακτικά, μη-αντιστρέψιμη. Ε, λοιπόν, τα δύο κλειδιά παράγονται από ειδικές συναρτήσεις, οι οποίες δέχονται σαν είσοδο έναν μεγάλο και *τυχαίο* αριθ-

μό. Αυτό που πρέπει να έχουμε στο μυαλό μας, είναι ότι όσο "περισσότερο τυχαίος" είναι ο αριθμός που δίνουμε στις συγκεκριμένες συναρτήσεις, τόσο πιο ασφαλή είναι και τα κλειδιά που παράγονται: Έχουν περισσότερο θόρυβο τυχαιότητας, όπως συνηθίζουμε να λέμε, γεγονός που δυσκολεύει ακόμα περισσότερο τον υπολογισμό του ιδιωτικού κλειδιού από το δημόσιο.

Η μεγάλη σημασία της τυχαιότητας αναδεικνύεται κι από τη συμπεριφορά ορισμένων γνωστών προγραμμάτων κρυπτογράφησης. Τα συγκεκριμένα προγράμματα δεν στηρίζονται αποκλειστικά στις δικές τους ρουτίνες για τον υπολογισμό τυχαίων αριθμών! Ζητούν από τον χρήστη να πραγματοποιήσει μια σειρά ενεργειών, όπως, ας πούμε, να κουνήσει το ποντίκι προς τυχαίες κατευθύνσεις, να πατήσει μερικά πλήκτρα στην τύχη κ.ο.κ., ώστε να συλλεγούν *πρόσθετα* τυχαία δείγματα.



Κατά την παραγωγή των κλειδιών κρυπτογράφησης, οι γνωστές εφαρμογές GnuPG και TrueCrypt ζητούν από το χρήστη να προβεί σε μερικές τυχαίες ενέργειες, όπως, για παράδειγμα, να κινήσει το ποντίκι προς τυχαίες κατευθύνσεις. Με αυτόν τον τρόπο, οι εφαρμογές λαμβάνουν δείγματα από μια διαδικασία που είναι πραγματικά τυχαία.

Φαινομενικά απρόβλεπτοι

Οι περισσότεροι από εμάς, στο πλαίσιο της εκμάθησης κάποιας γλώσσας προγραμματισμού έχουμε έρθει σε επαφή με τους τυχαίους ή ψευδοτυχαίους αριθμούς (pseudo random), όπως συνηθίζουμε να τους λέμε. Υποθέτουμε ότι το ακόλουθο προγραμματάκι σε C++ θα σας φανεί γνώριμο. Ακόμα κι αν δεν ασχοληθήκατε ποτέ με τη συγκεκριμένη γλώσσα, είναι σίγουρο ότι θα γράψατε κάτι παρόμοιο σε κάποια άλλη. Το μικρό μας πρόγραμμα χρησιμοποιεί τη συνάρτηση rand() και παράγει εκατό ψευδοτυχαίους αριθμούς από το διάστημα [0, 100].

```
#include <stdio.h>
#include <stdlib.h>
int main()
{
    int c, n;
    printf("Ten random numbers in [1,100]\n");
    for (c = 1; c <= 10; c++)
    {
       n = rand()%100 + 1;
       printf("%d\n", n);
    }
    return 0;
}
```

Όπως βλέπετε, οι "τυχαίοι" αριθμοί παράγονται με τη συνάρτηση rand(). Πίσω από αυτή τη συνάρτηση υπάρχει ένας μηχανισμός που παράγει αριθμούς κατά τέτοιον τρόπο, ώστε να είναι δύσκολο να προβλέψουμε κάθε φορά τον επόμενο. Είναι όμως στ' αλήθεια τυχαίοι, αυτοί οι αριθμοί; Προφανώς και όχι. Γι' αυτόν τον λόγο, εξάλλου, τους αποκαλούμε και ψευδοτυχαίους. Οι υπολογιστές αποτελούν αυτό που οι Μαθηματικοί ονομάζουν Ντετερμινιστικά Πεπερασμένα Αυτόματα (ΝΠΑ). Με απλά λόγια, αποτελούν "μηχανές" που μπορούν να βρεθούν σε ένα πεπερασμένο σύνολο καταστάσεων και, μάλιστα, η μετάβαση από τη μία κατάσταση στην άλλη πραγματοποιείται πάντα με κάποιον συγκεκριμένο και γνωστό τρόπο. Με ακόμα πιο απλά λόγια, από τη στιγμή που οι υπολογιστές λειτουργούν σύμφωνα με κάποιο πρόγραμμα, η συμπεριφορά τους είναι *πάντοτε* προβλέψιμη. Ας επιστρέψουμε στο θέμα μας, όμως. Εφόσον οι "τυχαίοι" αριθμοί παράγονται με κάποιον συγκεκριμένο αλγόριθμο, ανεξάρτητα από το αν έχουμε υπόψη μας αυτόν τον αλγόριθμο κι άσχετα με το αν θα μπορούσαμε να τον εκτελέσουμε στο νου μας, οι αριθμοί αυτοί είναι, τελικά, προβλέψιμοι.

Τα παραπάνω δεν σημαίνουν ότι η συνάρτηση rand() είναι άχρηστη. Σε ένα παιχνίδι, για παράδειγμα, είναι αρκετό το να μην μπορούμε να προβλέψουμε πότε θα εμφανιστεί ένας αντίπαλος, ακόμα κι αν αυτό το γεγονός δεν είναι πραγματικά τυχαίο :) Σε πολλές εφαρμογές, δηλαδή, δεν έχει καμία σημασία το αν οι αριθμοί που παράγονται από τη rand() είναι πραγματικά τυχαίοι — ή απλά τους αντιλαμβανόμαστε ως τέτοιους.

Όπως καταλαβαίνετε, αν και η αλγοριθμική παραγωγή τυχαίων αριθμών είναι θεωρητικά αδύνατη, το όλο ζήτημα συγκεντρώνει ιδιαίτερο ενδιαφέρον. Η παραγωγή μιας αλληλουχίας αριθμών, οι οποίοι θα παρουσιάζουν στατιστική συμπεριφορά όμοια μ' εκείνη των πραγματικά τυχαίων, αποτελεί σπουδαία πρόκληση. Βέβαια, ακόμη κι ένα τέτοιο σύνολο αριθμών, δεν είναι πραγματικά τυχαίο!

Διαχωριστικές γραμμές

Μέχρι στιγμής έχουμε αναφέρει τη λέξη "τυχαίος" πάρα πολλές φορές, εντός κι εκτός εισαγωγικών. Θέλουμε να πιστεύουμε ότι κάθε φορά γινόμασταν απόλυτα σαφείς και δεν προκαλούσαμε σύγχυση. Ωστόσο, νομίζουμε ότι έφτασε η ώρα να ξεκαθαρίσουμε λίγο τα πράγματα, δίνοντας κάποιους ορισμούς. Μια ακολουθία από bits θεωρείται ψευδοτυχαία, όταν ισχύουν τα ακόλουθα:

1. Ικανοποιεί το δόγμα της στατιστικής απαίτησης. Δηλαδή περνά με επιτυχία τους γνωστούς στατιστικούς ελέγχους, που έχουν αναπτυχθεί γι' αυτόν το σκοπό.

2. Η ακολουθία είναι απρόβλεπτη. Δηλαδή, όταν έχουμε στη διάθεσή μας ένα τμήμα της, είναι αδύνατο να μαντέψουμε το πώς συνεχίζεται.

Μα καλά, θα διαμαρτυρηθούν κάποιοι, τα παραπάνω κριτήρια δεν ισχύουν και για τους πραγματικά τυχαίους αριθμούς; Έτσι είναι! Οι πραγματικά τυχαίοι αριθμοί, όμως, έχουν κι ένα πρόσθετο γνώρισμα. Μια ακολουθία από bits είναι πραγματικά τυχαία, όταν ισχύουν όλα τα παραπάνω και, επιπρόσθετα, όταν είναι *αδύνατον* ν' αναπαραχθεί!

Πόσο εύκολο άραγε είναι να επαληθεύσουμε αυτό το τελευταίο κριτήριο; Πόσες φορές πρέπει να επαναλάβουμε μια διαδικασία, πριν αποφανθούμε για το αν παρουσιάζει περιοδικότητα ή όχι; Τελικά οι Μαθηματικοί, όπως και οι Πληροφορικοί, έχουν να λύσουν πολύ πιο ενδιαφέροντα προβλήματα από αυτά που μαθαίνουμε στο σχολείο.

Στο δεύτερο και τελευταίο άρθρο του μικρού μας αφιερώματος, που αρχίζει από τη σελίδα 62 του παρόντος τεύχους, εξετάζουμε μια μέθοδο παραγωγής ψευδοτυχαίων αριθμών, ενώ βλέπουμε και μερικές λύσεις για την παραγωγή πραγματικά τυχαίων αριθμών — στον υπολογιστή! Για την ώρα, θα σας προτείναμε να ρίξετε μια ματιά στο πολύ ενδιαφέρον βιβλίο με τίτλο "Handbook of Applied Cryptography" (http://cacr.uwaterloo.ca/hac). Στο κεφάλαιο 5, θα βρείτε και έναν ορισμό για τους ψευδοτυχαίους αριθμούς.

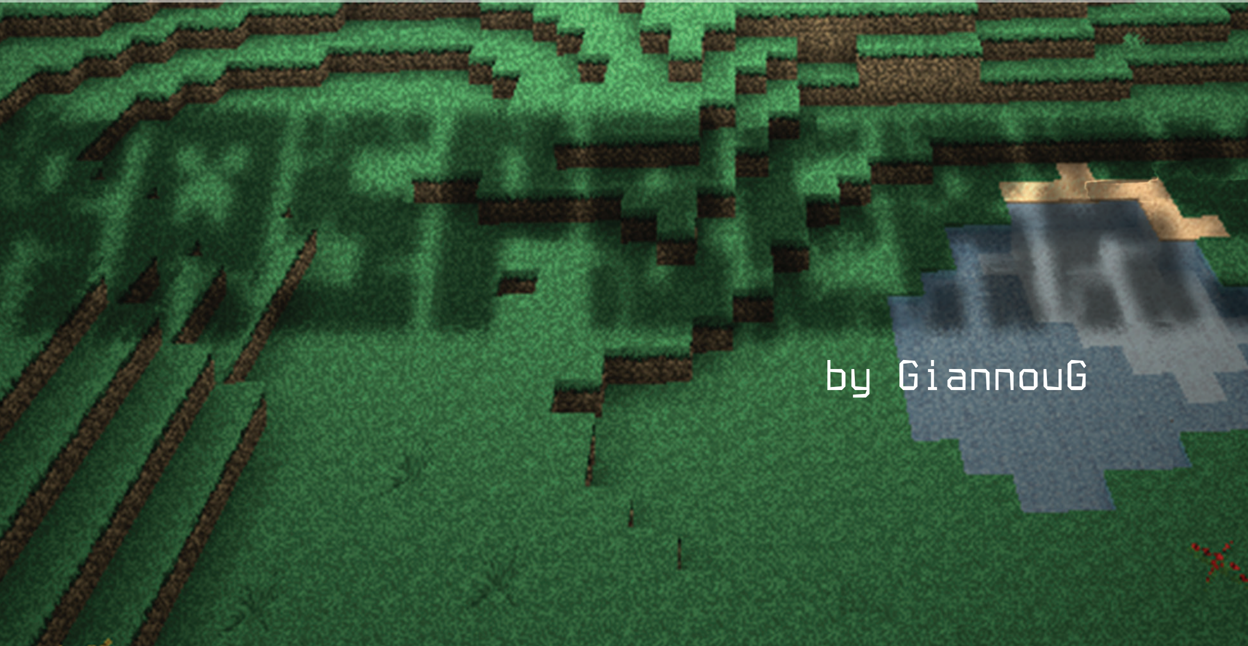
Skill: Beginner
Tags: Raspberry Pi, Raspbian, Minecraft, Python, LED



RASPBERRY

Από τα τουβλάκια στον πραγματικό κόσμο

Μας αρέσουν τα παιχνίδια, το ομολογούμε! Όχι, δεν καταλάβατε καλά. Δεν εννοούμε μόνο τα παιχνίδια που περιλαμβάνουν, π.χ., το στήσιμο ενός server και των σχετικών υπηρεσιών, το Arduino και τον προγραμματισμό. Μας αρέσουν *και* τα computer games. Αυτά που έχει στο νου της η μάνα μας κι ο πατέρας μας, όταν γκρινιάζουν. Εντάξει, δεν τρελαινόμαστε για όλους τους τίτλους που κυκλοφορούν, ούτε θα δηλώναμε "gamers". Αν βρούμε κάτι καλό, όμως, θα κολλήσουμε για τα καλά. Τα τελευταία τρία χρόνια, ας πούμε, έχουμε σκαλώσει με το Minecraft...



by GiannouG

Αν είσαστε από τα παιδιά που μεγάλωσαν με Lego, πρέπει να σας πούμε ότι το Minecraft αποτελεί τη φυσική συνέχεια αυτού του κλασσικού παιχνιδιού. Πέρα από τις απλές κατασκευές που μπορεί να φτιάξει κανείς, μπορεί να προσθέσει διάφορα mod στο παιχνίδι και να δημιουργήσει *οποιαδήποτε* κατασκευή επιθυμεί: Από απλές μηχανές και υπολογιστές που προγραμματίζονται σε LUA, μέχρι ατομικούς αντιδραστήρες[1]. Ας μην ξεφεύγουμε όμως, διότι γνωρίζετε πόσο εύκολα ενθουσιαζόμαστε. Αναρωτιέστε τώρα τι σχέση έχει το Minecraft με το περιοδικό μας; Αναρωτιέστε πώς τα συνδυάσαμε και τι σκοπεύουμε να κάνουμε;

Τι εννοεί ο ποιητής;

Πριν ξεκινήσουμε την ανάλυσή μας, πρέπει να σας πούμε ότι μας ενδιαφέρει μόνο το τεχνικό του μέρος του παιχνιδιού. Τα στοιχεία του gameplay, οι τρόποι επιβίωσης στον κόσμο του παιχνιδιού κι άλλα τέτοια, μας αφήνουν (για την ώρα τουλάχιστον) αδιάφορους και δεν θα τα αγγίξουμε σε αυτό το άρθρο[2]. (Νομίζατε ότι θα κάναμε ανάλυση στο παιχνίδι, ε; Χα!)

Το Minecraft, για όσους δεν το γνωρίζουν, είναι ένα παιχνίδι τύπου "sandbox". Ο χρήστης είναι περιορισμένος στο δικό του κόσμο κι εκεί είναι απόλυτα ελεύθερος να κάνει οτιδήποτε θέλει. Όλος ο κόσμος του Minecraft είναι φτιαγμένος από "κουτάκια" που ονομάζονται blocks. Οι δύο πιο σημαντικές ενέργειες που μπορεί να κάνει ο παίκτης, είναι να σπάσει (να καταστρέψει) ή να τοποθετήσει νέα blocks. Έτσι, μπορεί να αλλάξει τη δομή του κόσμου εντελώς. Μπορεί να χτίσει τεράστιες πόλεις και πολύπλοκες μηχανές — ή να τον γεμίσει με πυρηνικές βόμβες και να τα τινάξει όλα στον αέρα. Όπως είπαμε ήδη, ο κόσμος αποτελείται μόνο από blocks, τα οποία μπορεί να είναι από χώμα, πέτρα και διάφορα άλλα υλικά. Υπάρχουν επίσης κάποια "ειδικά" blocks, όπως οι διακόπτες και τα κουμπιά, τα tripwires και διάφοροι άλλοι "αισθητήρες", τους οποίους μπορούμε να ελέγξουμε μέσα από το παιχνίδι αλλά κι εκτός αυτού.

Minecraft και Raspberry Pi

Η Mojang, η εταιρία που δημιουργήθηκε για την ανάπτυξη του Minecraft, ως νέα και δραστήρια εταιρία στο χώρο της τεχνολογίας έχει κυκλοφορήσει μια ειδική έκδοση του παιχνιδιού αποκλειστικά για το Raspberry Pi. Δυστυχώς, η συγκεκριμένη έκδοση βασίζεται στο παλιό "Minecraft Pocket Edition", που υστερεί δραματικά σε σχέση με την κανονική έκδοση του παιχνιδιού για τον υπολογιστή. Αυτό όμως δεν

*Το Minecraft είναι από τα παιχνίδια που είτε δεν θα ασχοληθείς καθόλου είτε θα *λιώσεις* με τις ώρες. Η ελευθερία του sandbox σού επιτρέπει να δημιουργήσεις οτιδήποτε μπορείς να φανταστείς.*

1. Και φυσικά όλα αυτά συνδυάζονται και παίζουν αρμονικά μεταξύ τους ;)
2. Ή μήπως να το καθιερώναμε και να μιλούσαμε *και* για παιχνίδια; Ή και για ποδόσφαιρο! Εντάξει φτάνει. Ας ανοίξουμε ένα τερματικό, να συνέλθουμε.



Κάθε κουτάκι του κόσμου μπορεί να σπάσει και να τοποθετηθεί από το χρήστη. Αυτό επιτρέπει στους παίκτες να αλλάζουν τον κόσμο και να τον διαμορφώνουν όπως αυτοί θέλουν. Παρεμπιπτόντως, σημειώστε ότι ο κόσμος του παιχνιδιού παράγεται τυχαία και είναι εντελώς διαφορετικός σε κάθε "παρτίδα".

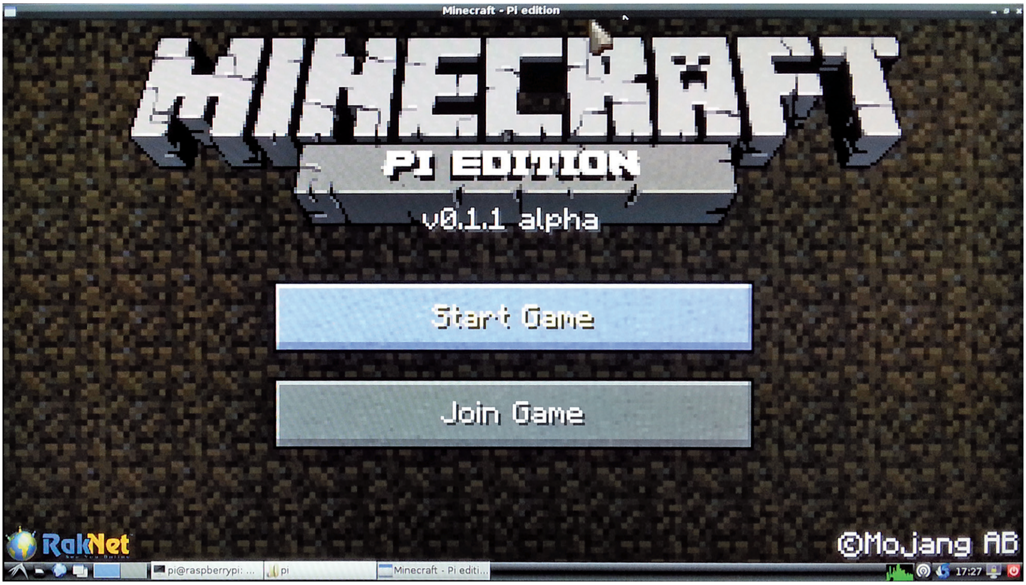
μας πειράζει καθόλου. Μπορεί να πρόκειται για προϊστορική έκδοση, η οποία μάλιστα έχει και μερικά bugs, αλλά είναι ιδανική για να φτιάξουμε ένα νέο interface για τις κατασκευές μας. Θα καταλάβετε σε λίγο που το πάμε.

Το ενδιαφέρον με το Minecraft Pi Edition, όπως ονομάζεται η έκδοση για το Raspberry Pi, βρίσκεται στο API που υποστηρίζει. Μέσα από μια θύρα TCP μπορούμε να επικοινωνούμε ελεύθερα με το παιχνίδι, να διαβάζουμε τιμές και να μεταβάλλουμε τον κόσμο όπως θέλουμε. Μπορούμε επίσης να μετακινούμε τον παίκτη και να διαβάζουμε τη θέση του. Με άλλα λόγια, μπορούμε να παίξουμε το παιχνίδι χρησιμοποιώντας τη θύρα TCP που έχει ορίσει η Mojang. Φυσικά εμείς δεν χρειάζεται να ασχοληθούμε σε τόσο χαμηλό επίπεδο, αφού υπάρχει ήδη μια βιβλιοθήκη που αξιοποιεί το API του παιχνιδιού και είναι γραμμένη σε Python.

Προετοιμασία εδάφους

Για να ξεκινήσουμε, θα χρειαστεί να εγκαταστήσουμε στο Raspberry Pi την τελευταία έκδοση του Raspbian. Αν προσπαθήσουμε να τρέξουμε το παιχνίδι σε παλιά (ή έστω λίγο παλιότερη) έκδοση του λειτουργικού, θα πάρουμε το μήνυμα ότι απουσιάζουν βασικές βιβλιοθήκες ή/και ότι το σύστημα δεν έχει πρόσβαση σε κάποια συσκευή. Αυτό το τονίζουμε για να σας προλάβουμε και να μην πάθετε κι εσείς τα ίδια.

Αφού λοιπόν κατεβάσουμε την τελευταία έκδοση του λειτουργικού από το `http://www.raspberrypi.org/downloads`, μπορούμε να την περάσουμε στην κάρτα SD του Raspberry Pi χρησιμοποιώντας το Win32DiskImager. Τα βήματα είναι γνωστά σε



Η Mojang κυκλοφόρησε το Minecraft Pi Edition αποκλειστικά για το Raspberry Pi. Πρόκειται για μια μεταφορά του Raspberry Pi για κινητά.

όλους τους κατόχους του Raspberry Pi και δεν χρειάζεται να γίνουμε κουραστικοί, επαναλαμβάνοντας τα ίδια και τα ίδια. Για τις δοκιμές μας εξάλλου δουλέψαμε στο σαλόνι κι όχι στο "battle station", από όπου παίζουμε συνήθως τα παιχνίδια μας. Αυτό το κάναμε γιατί χρειαζόμασταν ένα monitor με HDMI (η τηλεόραση στο σαλόνι). Χρησιμοποιήσαμε επίσης ένα ασύρματο πληκτρολόγιο και ποντίκι, καθώς κι ένα USB WiFi module για την σύνδεση του Raspberry στο δίκτυο του σπιτιού και στο Internet. Σημειώστε ότι όλες τις εντολές που θα δούμε στη συνέχεια τις εκτελέσαμε στο τερματικό LxTerminal FTW. Θα μπορούσαμε όμως να είχαμε χρησιμοποιήσει και μια σύνδεση SSH, από άλλον υπολογιστή.

Ήρθε η ώρα να προσθέσουμε το Minecraft Pi Edition στο Raspberry Pi. Όλες οι απαραίτητες βιβλιοθήκες και τα εργαλεία υπάρχουν εξαρχής στο Raspbian. Το μόνο που χρειάζεται να κάνουμε εμείς είναι να κατεβάσουμε το παιχνίδι και να αποσυμπιέσουμε το σχετικό πακέτο:

```
cd ~
wget \ https://s3.amazonaws.com/assets.minecraft.net/pi/minecraft-pi-
0.1.1.tar.gz
tar -zxvf minecraft-pi-0.1.1.tar.gz
```

Η αποσυμπίεση θα δημιουργήσει έναν νέο κατάλογο, μέσα στον οποίο υπάρχει το εκτελέσιμο αρχείο που ξεκινάει το παιχνίδι:

```
cd mcpi/
./minecraft-pi
```

Μπορούμε να ασχοληθούμε ελεύθερα για ένα τέταρτο με το παιχνίδι, αλλά προσέξτε μην κολλήσετε. (Ο κύριος που πατάει αυτή τη στιγμή τα πλήκτρα, κόλλησε.) Ιδιαίτερη εντύπωση μας έκανε η ταχύτητα του παιχνιδιού, που σε σύγκριση με άλλες

εφαρμογές του Raspberry Pi πρέπει να πούμε ότι πετάει. Αφού χτίσουμε κάτι στα γρήγορα —ένα σπίτι, μια έπαυλη ή ότι άλλο θελήσουμε— μπορούμε να προχωρήσουμε σε πιο ενδιαφέροντα πράγματα.

Python For The Win!

Το πρωτόκολλο του παιχνιδιού είναι αρκετά απλό και, για καλή μας τύχη, έχει υλοποιηθεί σε δύο βιβλιοθήκες για την Python και τη Java. Εμείς, όπως αντιλαμβάνεστε όσοι μας ξέρετε λίγο, ενδιαφερόμαστε για τη βιβλιοθήκη της Python, αφού η γλώσσα είναι ήδη εγκατεστημένη στο Raspbian και κυρίως επειδή την αγαπάμε ιδιαιτέρως. Στο αρχείο mcpi_protocol_spec.txt, που βρίσκεται μέσα στον κατάλογο του παιχνιδιού, υπάρχει μια σύντομη περιγραφή του πρωτοκόλλου. Ενδιαφερόμαστε για συγκεκριμένες λειτουργίες και κυρίως για την κίνηση του παίκτη και την αλληλεπίδρασή του με τον κόσμο του παιχνιδιού. Ας δούμε μερικές από αυτές.

```
chat.post(message)
```

Στέλνει στο παιχνίδι ένα μήνυμα, το οποίο εμφανίζεται στην περιοχή chat. Αυτή η λειτουργία είναι χρήσιμη στις περιπτώσεις που θέλουμε να εμφανιστεί κάποιο μήνυμα στον παίκτη. Στην περίπτωσή μας, φάνηκε χρήσιμο στο να υπολογίζουμε πόση απόσταση έκανε ο παίκτης[3].

```
player.getPos()
```

Επιστρέφει την ακριβή θέση του παίκτη στους 3 άξονες του κόσμου (X, Y και Z). Οι συντεταγμένες που επιστρέφονται είναι δεκαδικοί κι αφορούν το εκάστοτε block του κόσμου, στο οποίο στέκεται ο παίκτης. Οι άξονες X και Z βρίσκονται στο "επίπεδο" του κόσμου, ενώ ο Y περιγράφει το ύψος.

```
world.getBlock(x, y, z)
```

Επιστρέφει το ID του block, στις συντεταγμένες που ορίσουμε. Προσοχή: Δεν επιστρέφει το "data ID", που θα μας χρησίμευε στην περίπτωση ενός διακόπτη, ώστε να καταλάβουμε αν είναι πατημένος ή όχι. Το ID περιγράφει τον τύπο ενός block, ενώ το "data ID" περιγράφει την κατάστασή του (ή και άλλες ιδιότητες).

```
world.setBlock(x, y, z, id)
```

Τοποθετεί ένα block στις συντεταγμένες που ορίζουμε και του αποδίδει ένα συγκεκριμένο ID.

```
events.block.hits()
```

Επιστρέφει τη λίστα με τα "block hit events" που συνέβησαν στον κόσμο. Όταν ο

3. Πριν αποφασίσουμε να τα ξηλώσουμε όλα για να το κάνουμε απλούστερα σε έναν άξονα — περισσότερα σε λίγο όμως.

παίκτης χτυπήσει/αγγίξει ένα block, τότε η συγκεκριμένη κίνηση, μαζί με επιπλέον πληροφορίες όπως το id του block, καταγράφονται σε αυτή τη λίστα.

Χρησιμοποιώντας τις παραπάνω εντολές μπορούμε να γράψουμε προγράμματα σε Python που θα λαμβάνουν δεδομένα από το παιχνίδι. Έτσι, είμαστε σε θέση να ελέγχουμε ένα τέτοιο πρόγραμμα, παίζοντας απλά το παιχνίδι. Κατασκευάσαμε ένα απλό πρόγραμμα, το οποίο αναβοσβήνει ένα LED σύμφωνα με τη θέση του παίκτη μέσα στον κόσμο του Minecraft. Ας το δούμε.



*Για να 'μαστε ειλικρινείς, δεν πιστεύουμε ότι το παιχνίδι κυκλοφόρησε ως *παιχνίδι*, αλλά περισσότερο ως ένα εργαλείο πειραματισμών με το Raspberry Pi :)*

Η δική μας υλοποίηση

Αρχικά, θα χρειαστεί να αντιγράψουμε ολόκληρη τη βιβλιοθήκη από τον σχετικό κατάλογο του παιχνιδιού. Μιλάμε για τη βιβλιοθήκη της Python που αξιοποιεί το API του παιχνιδιού. Για την ανάπτυξη του προγράμματός μας, δημιουργήσαμε έναν ξεχωριστό κατάλογο και του δώσαμε το εμπνευσμένο όνομα… Minecraft. Έτσι, ξεκινήσαμε την εργασία μας γράφοντας τα ακόλουθα:

```
mkdir ~/Minecraft
cp -r ~/mcpi/api/python/mcpi ~/Minecraft
```

Αφού αντιγράψουμε τη βιβλιοθήκη μπορούμε να ξεκινήσουμε το χτίσιμο της εφαρ-

μογής μας. Προφανώς, στην αρχή του προγράμματος αναφέρουμε τις απαραίτητες βιβλιοθήκες:

```
import time
import os
import mcpi.minecraft as minecraft
import mcpi.block as block
import RPi.GPIO as GPIO
```

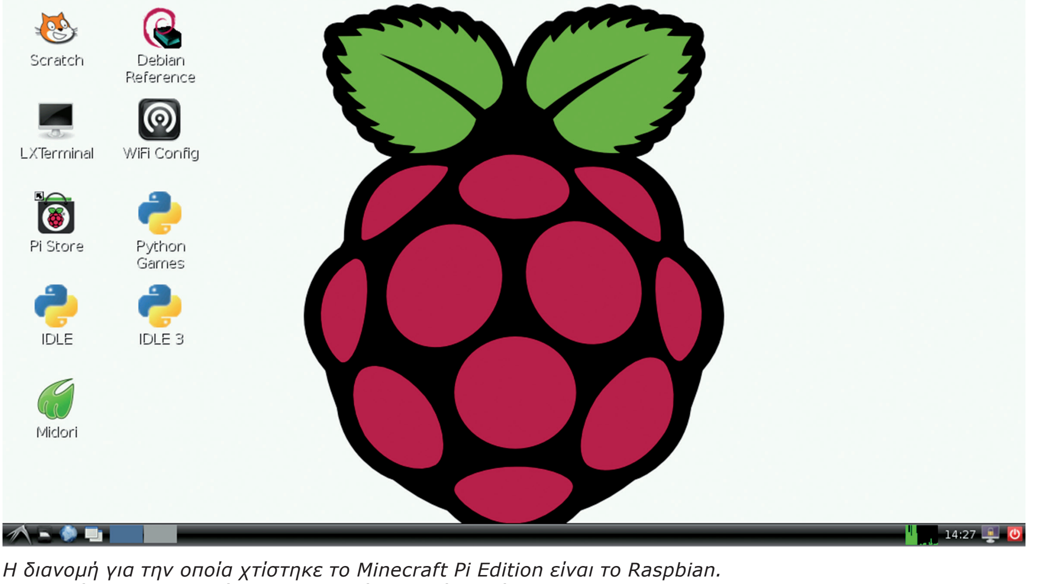
Οι βιβλιοθήκες time και os είναι βασικές της γλώσσας. Εμείς τις χρειαζόμαστε για εντολές όπως η sleep. Για τη διασύνδεση με το Minecraft, χρειαζόμαστε τόσο την κύρια βιβλιοθήκη (minecraft), όσο κι εκείνη που περιγράφει τα διάφορα blocks (block). Τέλος, ενσωματώνουμε τη βιβλιοθήκη GPIO, για τον εύκολο έλεγχο των ακροδεκτών IO του Raspberry.

```
GPIO.setmode(GPIO.BCM)
GPIO.setup(24, GPIO.OUT)
```

Θυμόσαστε το άρθρο "Μετατρέψτε το Raspberry Pi σε Arduino!" από το τεύχος 017; Τότε, είχαμε μιλήσει για το πώς μπορούμε να ελέγξουμε την κατάσταση των ακροδεκτών του Raspberry Pi και είχαμε αναφερθεί εκτενώς στη βιβλιοθήκη GPIO. Θα σας προτείναμε να του ξαναρίξετε μια ματιά. Στις παραπάνω γραμμές, πάντως, δηλώνουμε ότι θα χρησιμοποιήσουμε τον ακροδέκτη 24 ως έξοδο.

```
playerPos = mc.player.getPos()
playerPos = minecraft.Vec3(int(playerPos.x), int(playerPos.y),
int(playerPos.z))
```

Το κυρίως πρόγραμμα ξεκινά διαβάζοντας τη θέση του παίκτη μέσα στον κόσμο του παιχνιδιού. Αρχικά, διαβάζουμε τη θέση με την εντολή player.getPos, που αναφέραμε προηγουμένως. Στη συνέχεια μετατρέπουμε τις συντεταγμένες στη μορφή "Vec3", την οποία χρησιμοποιεί κι αναγνωρίζει η μηχανή του παιχνιδιού.



Η διανομή για την οποία χτίστηκε το Minecraft Pi Edition είναι το Raspbian. Προσοχή: Θα χρειαστούμε την πιο πρόσφατη έκδοση του.

```
blockx = playerPos.x + 5
blocky = playerPos.y - 1
blockz = playerPos.z
mc.setBlock(blockx, blocky, blockz, block.TNT)
```

Το πρόγραμμά μας θα ανάβει ένα LED, όταν ο χρήστης πατήσει σε ένα συγκεκριμένο block. Ποιο θα είναι αυτό το block, όμως; Στην αρχή θελήσαμε να ορίσουμε μια κυκλική περιοχή, με ακτίνα 5 blocks και κέντρο την αρχική θέση του παίκτη. Ωστόσο δεν υπήρχε κανένας λόγος να περιπλέξουμε τόσο την πρώτη μας εφαρμογή και σκεφτήκαμε να αποφύγουμε τα διανύσματα. Έτσι, καταλήξαμε στην επιλογή ενός συγκεκριμένου μπλοκ, πέντε κουτάκια μπροστά από την αρχική θέση του παίκτη. Στις παραπάνω γραμμές, υπολογίζουμε πρώτα τη θέση του block που θα λειτουργεί ως διακόπτης. Για τη θέση στον άξονα Y αφαιρούμε μια μονάδα από την αντίστοιχη θέση του παίκτη, ώστε το block-διακόπτης να βρίσκεται στο επίπεδο που πατάνε τα πόδια του κι όχι μπροστά τους. Μετά δημιουργούμε ένα νέο block κι επιλέγουμε τον τύπο TNT, ώστε όταν παίζουμε να το ξεχωρίζουμε εύκολα.

```
while (True):
    GPIO.output(24, False)

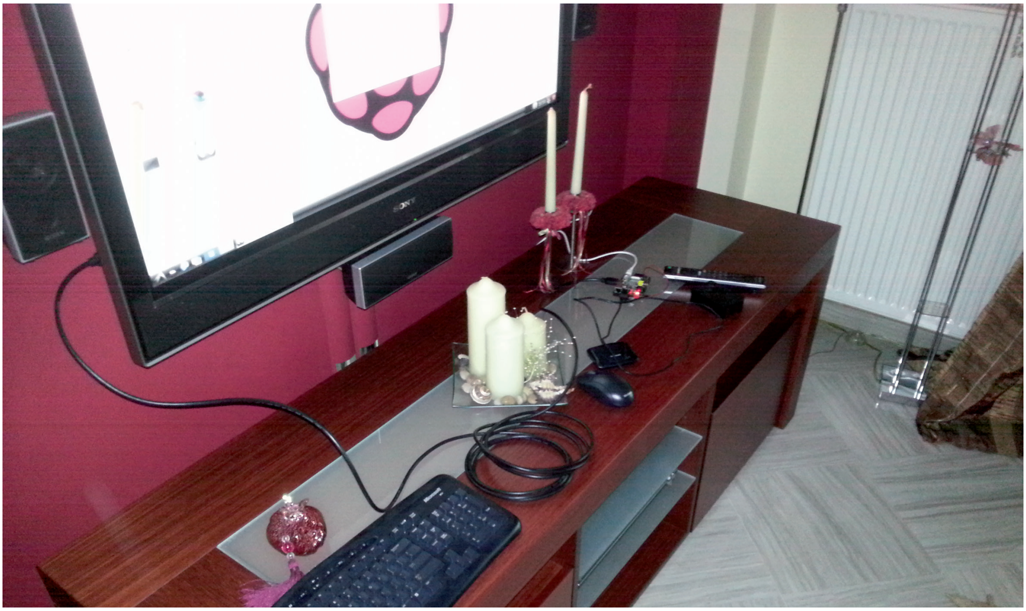
    playerPos = mc.player.getPos()

    playerPos = minecraft.Vec3( \
        int(playerPos.x), int(playerPos.y), int(playerPos.z))

        if (playerPos.x == blockx):

                GPIO.output(24, True)

        time.sleep(2)
```



Όταν δεν έχεις διαθέσιμο οθόνη με HDMI, αναγκάζεσαι να μετακομίσεις κοντά στην τηλεόραση και κατά πάσα πιθανότητα στο σαλόνι. Κάπως έτσι, καταλήξαμε να παίζουμε Minecraft σε μια οθόνη full HD 46 ιντσών.



Για την επικοινωνία του Raspberry Pi με το οικιακό δίκτυο και με το Internet, χρειαστήκαμε ένα USB WiFi. Ευτυχώς, το Raspbian διαθέτει ειδικό εργαλείο για τη ρύθμιση αυτών των συσκευών.

Ο κύριος βρόχος του προγράμματός μας. Σε κάθε επανάληψη, θέτουμε τον ακροδέκτη 24 στην κατάσταση FALSE κι ελέγχουμε αν ο παίκτης στέκεται πάνω στο ειδικό block που δημιουργήσαμε προηγουμένως. Εφόσον συμβαίνει κάτι τέτοιο, αλλάζουμε την κατάσταση του ακροδέκτη σε TRUE.

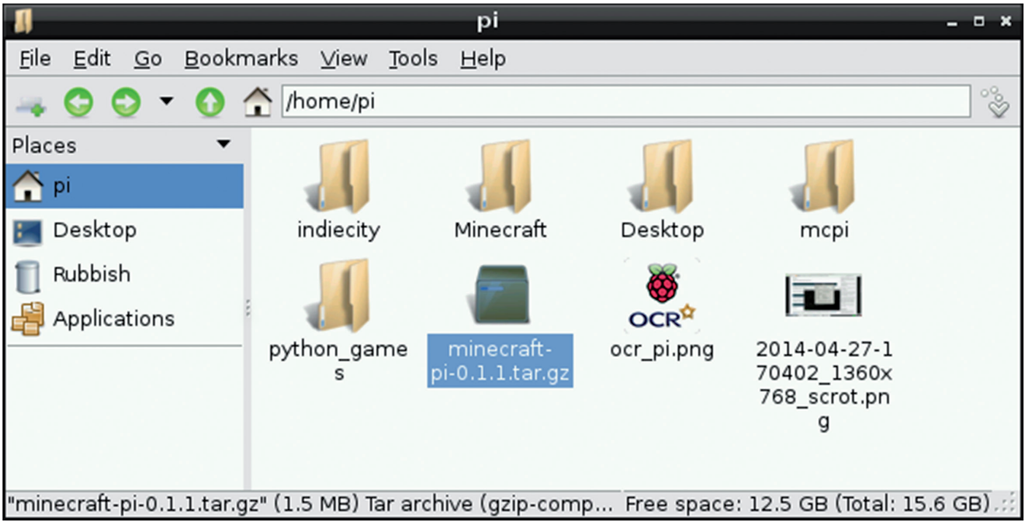
Όπως μπορείτε να φανταστείτε, στον ακροδέκτη 24 πρέπει να συνδέσουμε ένα LED. Συγκεκριμένα, πρέπει να συνδέσουμε την άνοδο του LED στον εν λόγω ακροδέκτη καθώς και την κάθοδο στο GND της πλακέτας του Raspberry. Εδώ πρέπει να σημειώσουμε ότι οι ακροδέκτες του Raspberry Pi δεν μπορούν να προσφέρουν παραπάνω από 16mA ρεύματος. Αν προσπαθήσουμε να τραβήξουμε περισσότερο, υπάρχει ο κίνδυνος να κάψουμε τον ακροδέκτη ή ακόμα κι ολόκληρη την πλακέτα! Επομένως, πριν συνδέσουμε το LED θα ήταν φρόνιμο να τσεκάρουμε πόσο ρεύμα τραβάει. Αν αυτό το μέγεθος υπερβαίνει τα 16mA που μπορεί να δώσει το Raspberry, πρέπει να προσθέσουμε οπωσδήποτε και μια αντίσταση.

Περιττό να πούμε ότι το απλούστατο προγραμματάκι μας δούλεψε αμέσως. Το τρέξαμε *αφού* ξεκινήσαμε το παιχνίδι και, κάθε φορά που παίκτης πατούσε στο block-διακόπτη, το LED άναβε. Όταν ο παίκτης πατούσε οπουδήποτε άλλου, το LED έσβηνε. Μπορεί να μιλάμε για μια πολύ απλά λειτουργία, αλλά εμάς μας εντυπωσίασε. Η κατασκευή μας λειτουργούσε σαν μια γέφυρα επικοινωνίας μεταξύ του πραγματικού κόσμου και του Matrix! (Είπαμε: Εντυπωσιαστήκαμε :D)

Ατέλειες...

Δυστυχώς, η Mojang δεν δείχνει ιδιαίτερο ενδιαφέρον για το Minecraft Pi Edition. Μετά την αρχική κυκλοφορία βγήκαν μόλις τρεις αναβαθμισμένες εκδόσεις κι από τότε δεν έχει ακουστεί τίποτα για το μέλλον του παιχνιδιού. Αυτό σημαίνει ότι θα είμαστε περιορισμένοι στη συγκεκριμένη έκδοση — ίσως για πάντα.

Ένα πρόβλημα που συναντήσαμε αφορά στα events του παιχνιδιού και στον τρόπο που τα διαχειρίζεται η βιβλιοθήκη της Python. Πριν φτιάξουμε την εκδοχή του προγράμματος που παρουσιάσαμε, σκεφτόμασταν κάτι διαφορετικό: Θέλαμε να ελέγχουμε το πραγματικό LED με ένα block-διακόπτη του κόσμου του Minecraft. Αυτό θα μπορούσαμε να το πετύχουμε περιμένοντας ένα event "χτυπήματος" του παίκτη, πάνω σε ένα block με ID διακόπτη. Στη συνέχεια, θα διαβάζαμε το data ID του συγκεκριμένου block κι από εκεί να θα μαθαίναμε την κατάσταση του διακόπτη. Σύμφωνα με αυτή την τιμή, θα τροποποιούσαμε κατάλληλα και την κατάσταση του LED. Δυστυχώς, αυτή την παραλλαγή του προγράμματος δεν καταφέραμε να την υλοποιήσουμε ποτέ. Μάλιστα, ψάχνοντας για κάποια λύση σε διάφορα forum, διαπιστώσαμε ότι το πρόβλημα με τη διαχείριση των events είναι ιδιαίτερα διαδεδομένο και δεν υπάρχει γενική λύση.



Αφού κατεβάσουμε το Minecraft Pi Edition, μπορούμε να το αποσυμπιέσουμε και να το τρέξουμε για μια πρώτη δοκιμή... λίγων ωρών.

Κλείνοντας

Πριν κλείσουμε, αξίζει ν' αναφέρουμε ότι μπορούμε να έχουμε στη διάθεσή μας όλη τη λειτουργικότητα του API και ταυτόχρονα να απολαμβάνουμε την πιο πρόσφατη και πλήρη έκδοση του παιχνιδιού. Για το σκοπό αυτό θα χρειαστεί να εργαστούμε σε PC, στο οποίο πρέπει να σηκώσουμε έναν server Bukkit και να του προσθέσουμε το plugin RaspberryJuice (`http://dev.bukkit.org/bukkit-plugins/raspberryjuice`). Όλα αυτά, όμως, ενδέχεται να μας απασχολήσουν σε κάποιο μελλοντικό άρθρο.

Phyical / Raspberry Pi Name			
3.3v	1	2	5v
SDA0	3	4	DNC
SCL0	5	6	0v
GPIO 7	7	8	TX
DNC	9	10	RX
GPIO 0	11	12	GPIO 1
GPIO 2	13	14	DNC
GPIO 3	15	16	GPIO 4
DNC	17	18	GPIO 5
SPI MOSI	19	20	DNC
SPI MISO	21	22	GPIO 6
SPI SCLK	23	24	SP10 CE0 N
DNC	26	26	SP10 CE1 N

Broadcom names			
3.3v	1	2	5v
I2C0 SDA	3	4	DNC
I2C0 SCL	5	6	0v
GPIO 4	7	8	UART TXD
DNC	9	10	UART RXD
GPIO 17	11	12	GPIO 18
GPIO 21	13	14	DNC
GPIO 22	15	16	GPIO 23
DNC	17	18	GPIO 24
SPI MOSI	19	20	DNC
SPI MISO	21	22	GPIO 25
SPI SCLK	23	24	SP10 CE0 N
DNC	26	26	SP10 CE1 N

Δείτε το pin-out του Raspberry Pi, αν και είμαστε σίγουροι ότι θυμόσαστε τα πάντα από το σχετικό άρθρου του deltaHacker 017.

Για την ώρα, μόνο και μόνο για να εξάψουμε τη φαντασία σας, θα αναφέρουμε μερικές ιδέες μας. Πώς θα σας φαινόταν αν κατασκευάζαμε μια μακέτα του σπιτιού μας στο Minecraft και ελέγχαμε όλα τα *πραγματικά* φώτα, παίζοντας το παιχνίδι; Φυσικά, εφόσον χρησιμοποιούμε την Python, τίποτα δεν μας περιορίζει στους διακόπτες και στα LED. Θα μπορούσαμε εξίσου εύκολα να ελέγξουμε τη λειτουργία μιας υπηρεσίας σε έναν server, να ενεργοποιήσουμε ένα πρόγραμμα λήψης backup, να ξεκινήσουμε την αναπαραγωγή ενός τραγουδιού κ.ο.κ. Η σύνδεση του κόσμου του παιχνιδιού με τον πραγματικό έχει τουλάχιστον πλάκα, ενώ αν είναι κανείς ιδιαίτερα ευρηματικός, μπορεί να αποδειχθεί και χρήσιμη. (Μπρρρ, πολύ Matrix θυμίζει η υπόθεση.)

Skill: Beginner
Tags: Heartbleed, OpenSSL, heartbeat, buffer overrun, data validation



by Sp1r@lEv0lut10n

Προσεγγίζοντας θετικά την καταστροφή!

Από τα λαμπάκια του ρούτερ, σε μία από τις σημαντικότερες ευπάθειες του διαδικτύου. Ένα παιδαριώδες σφάλμα στον κώδικα του OpenSSL, καθιστά κάθε σενάριο του κινηματογράφου ρεαλιστικό. Την ίδια στιγμή, η παρουσία του συγκεκριμένου σφάλματος στον κώδικα μιας τόσο κρίσιμης εφαρμογής, ευνοεί την ανάπτυξη πρόσθετων σκοτεινών σεναρίων. Υπάρχει τίποτα χρήσιμο πίσω από όλον αυτό τον κουρνιαχτό;

...Τα αρπίσματα στο πιάνο δίνουν και παίρνουν. Το χέρι βγαίνει διστακτικά από τα σκεπάσματα, σκαρφαλώνει στο κομοδίνο και ψηλαφίζει την επιφάνεια, αναζητώντας το ξυπνητήρι. Το βρίσκει. Λίγο ακόμα σκαρφάλωμα κι η παλάμη καταφέρνει να βρει το μεγάλο οβάλ κουμπάκι. Το ξυπνητήρι αναπηδάει από τη δύναμη που δέχεται, αλλά τα αρπίσματα δεν σταματούν. Δεν ήταν το ξυπνητήρι, τελικά. Ήταν ο νέος ήχος κλήσης που είχε επιλέξει χτες το απόγευμα επειδή του φαινόταν ποιοτικός, ο οποίος τώρα ήταν απλά εξοργιστικός. Το χέρι βρίσκει τη συσκευή και τη μετακινεί κάπου πάνω από το κεφάλι. Τα μάτια ανοίγουν με δυσκολία και προσπαθούν να εστιάσουν στην εκτυφλωτική οθόνη. "Δεν το πιστεύω", σκέφτεται ο Νικόλας. Είναι ο νέος συνάδελφος από τη δουλειά. "Αφού του είπα, του @#$%, να μη με πάρει τηλέφωνο εκτός κι αν αρπάξουν φωτιά οι σέρβερ!". Απαντά στην κλήση και μουγκρίζει κάτι ανάμεσα στο "τι θες;" και στο "αυτό θα το μετανιώσεις" — αν και δεν χρησιμοποιεί καμία από αυτές τις λέξεις. Λίγα δευτερόλεπτα μετά, ο εκνευρισμός προς το συνάδελφο αντικαθίσταται από έναν παραλυτικό πανικό. "Μακάρι να είχαν πάρει φωτιά", σκέφτεται. "Μα, γιατί δεν πήραν φωτιά;" Η σκέψη του Νικόλα κάνει άσκοπους κύκλους κι αποφεύγει την ουσία: Πόσοι το γνώριζαν αυτό; Μήπως πρόλαβαν να το εκμεταλλευτούν; Πώς θα το εξηγήσω στο διευθυντή που είναι άσχετος με τους υπολογιστές και τα δίκτυα;

Κάτι τέτοιο φανταζόμαστε να συνέβη σε διαχειριστές τραπεζικών κι άλλων κρίσιμων συστημάτων, όταν ανακοινώθηκε μια ευπάθεια στο OpenSSL που έγινε γνωστή ως Heartbleed:

```
 OpenSSL Security Advisory [07 Apr 2014]
========================================

TLS heartbeat read overrun (CVE-2014-0160)
==========================================

A missing bounds check in the handling of the TLS heartbeat
extension can be used to reveal up to 64k of memory to a connected
client or server.

Only 1.0.1 and 1.0.2-beta releases of OpenSSL are affected
including 1.0.1f and 1.0.2-beta1.

Thanks for Neel Mehta of Google Security for discovering this bug
and to Adam Langley <agl@chromium.org> and Bodo Moeller <bmoeller@
acm.org> for preparing the fix.

Affected users should upgrade to OpenSSL 1.0.1g. Users unable
to immediately upgrade can alternatively recompile OpenSSL with
-DOPENSSL_NO_HEARTBEATS.

1.0.2 will be fixed in 1.0.2-beta2.
```

Μήπως αναρωτιέστε για την ώρα δημοσίευσης του παραπάνω μηνύματος; Σκεφτείτε ότι μιλάμε για ένα bug σε ένα σύστημα που χρησιμοποιείται σε μια τεράστια μερίδα των web servers του πλανήτη, για τη διαχείριση ασφαλών συνδέσεων HTTPS. Όλο και κάπου θα ήταν ξημερώματα — και μάλιστα ανεξαρτήτως της σοβαρότητας του προβλήματος :P

Ολέθρια ευπάθεια

Ας σοβαρευτούμε, όμως. Το OpenSSL αποτελεί την πλέον διαδεδομένη υλοποίηση των πρωτοκόλλων TLS/SSL. Πρόκειται για τη μηχανή που αναλαμβάνει τη δημιουργία των κλειδιών κρυπτογράφησης, την ίδια την κρυπτογράφηση και, γενικότερα, τη διαχείριση των κρυπτογραφημένων συνδέσεων HTTPS στο web. Όπως δηλώνει και το όνομά του, το OpenSSL αποτελεί προϊόν Ανοιχτού Λογισμικού και αξίζει να σημειώσουμε ότι χρησιμοποιείται σε μερικά εκατομμύρια web servers. Σε αυτό ακριβώς το σύστημα, λοιπόν, εντοπίστηκε μια εξαιρετικά επικίνδυνη ευπάθεια, η οποία ονομάστηκε "Heartbleed". Αξιοποιώντας την, ένας επιτιθέμενος μπορεί να κλέψει ακόμα και τα κλειδιά κρυπτογράφησης μιας συνεδρίας. Μετά από αυτό, η όποια μυστικότητα της σύνδεσης καταλύεται ολοκληρωτικά. Μάλιστα, αν ο επιτιθέμενος έχει προνοήσει κι έχει καταγράψει όλα τα κρυπτογραφημένα δεδομένα της συνεδρίας, θα μπορέσει να αποκρυπτογραφήσει ακόμα και τα στοιχεία σύνδεσης (username και password) που εξέπεμψε ο χρήστης. Σκεφτείτε τώρα την περίπτωση που η υπηρεσία με την ευάλωτη εκδοχή του OpenSSL, είναι το web interface μιας τράπεζας. Ο επιτιθέμενος θα μπορεί να συνδεθεί με τα στοιχεία του ανυποψίαστου πελάτη-θύματος και να του χαρίσει ένα ταξίδι στην κόλαση. Εν ολίγοις, το Heartbleed επιτρέπει την πραγματοποίηση όλων των σεναρίων που έχουμε παρακολουθήσει κατά καιρούς στον κινηματογράφο: Από τη σύνδεση σε μια κρατική υπηρεσία για την υποκλοπή μυστικών που μπορούν να προκαλέσουν ακόμη και πόλεμο, μέχρι τη σύνδεση στον server μιας σχολής, για την αλλοίωση της βαθμολογίας (για κάποιους, αυτό είναι πιο σημαντικό κι από την πρόκληση πολέμου :)) Μάλιστα, η σχετική επίθεση μπορεί να πραγματοποιηθεί εύκολα, γρήγορα και χωρίς να αφήσει το παραμικρό ίχνος. Όπως ήταν αναμενόμενο, η δημοσίευση αυτής της ευπάθειας έκανε μεγάλο ντόρο, διαδόθηκε αστραπιαία και τροφοδότησε διάφορα σενάρια συνωμοσίας.

Ψυχραιμία

Όπως αντιλαμβάνεστε, εμείς δεν πρόκειται να καταπιαστούμε με τέτοια σενάρια, αναζητώντας προθέσεις μυστικών υπηρεσιών. Όχι ότι μπορούμε να αποκλείσουμε αυτά τα σενάρια —κάθε άλλο, μάλιστα—, αλλά πιστεύουμε ότι μπορούμε να κάνουμε κάτι πολύ πιο εποικοδομητικό. Στο παρόν άρθρο, λοιπόν, θα μελετήσουμε την εν λόγω ευπάθεια και με την ευκαιρία θα φωτίσουμε το ηθικό δίδαγμα που κατά την ταπεινή μας γνώμη προκύπτει από την όλη ιστορία. Στο σημείο αυτό, πάντως, για την αποφυγή οποιουδήποτε μπερδέματος οφείλουμε να διευκρινίσουμε ότι η ευπάθεια Heartbleed δεν αφορά στις συνδέσεις SSH. Το πλέον διαδεδομένο πακέτο για την υλοποίηση αυτών των συνδέσεων είναι το OpenSSH, το οποίο δεν έχει καμία σχέση με το OpenSSL. Το Heartbleed αφορά κυρίως στις συνδέσεις HTTPS (αλλά όχι μόνο). Στη συνέχεια, θα εξετάσουμε το ευπαθές τμήμα του κώδικα του OpenSSL αλλά, πριν από αυτό, θα κάνουμε την απολύτως απαραίτητη θεωρητική εισαγωγή.

Επιβεβαίωση λειτουργίας

Έχετε παρατηρήσει ποτέ το μικρό LED της πλακέτας του Arduino, κατά τη μεταφορά ενός προγράμματος; Αναβοσβήνει. Το ίδιο συμβαίνει και με το LED ενός ενεργοποιημένου συναγερμού, μιας κάμερας κατά τη διάρκεια της εγγραφής, του modem/router όταν έχει συνδεθεί στο Internet και πάει λέγοντας. Αυτά τα LED έχουν σαν αποστολή να ενημερώνουν το χρήστη για την ομαλή εξέλιξη της εκάστοτε λειτουργίας. Ε, λοιπόν, παρόμοιους μηχανισμούς ενσωματώνουν και πρωτόκολλα που χρησιμοποιούνται στο διαδίκτυο, όπως είναι το TLS που βρίσκεται πίσω από τις ασφα-

λείς συνδέσεις HTTPS. Ο μηχανισμός που μας ενδιαφέρει ονομάζεται "heartbeat" και προβλέπει την αυτόματη αποστολή ενός μικρού πακέτου δεδομένων, σε τακτά χρονικά διαστήματα. Το heartbeat έχει σαν αποστολή να ενημερώνει το πρόγραμμα (client application) του χρήστη μιας υπηρεσίας, για την κατάσταση της σύνδεσης. Σε πολλές περιπτώσεις, εκτός από την αυτοματοποιημένη λειτουργία, ο ίδιος μηχανισμός μπορεί να ενεργοποιηθεί *οποιαδήποτε* στιγμή το ζητήσει ο χρήστης, με τον ίδιο πάντα σκοπό: Την επιβεβαίωση ότι ο server και η σχετική υπηρεσία εξακολουθούν να λειτουργούν κανονικά. Ας περάσουμε τώρα στο OpenSSL.

Μοιραία παράλειψη

Το OpenSSL δεν αποτελεί εξαίρεση. Από τα τέλη του 2011 διαθέτει έναν παρόμοιο μηχανισμό, ο οποίος μάλιστα έχει και το καθιερωμένο όνομα (heartbeat). Ο εν λόγω μηχανισμός επιτρέπει σε έναν χρήστη να στείλει στον server ένα σύντομο μήνυμα και να ζητήσει σαν απάντηση ένα τμήμα αυτού του μηνύματος. Αν ο server απαντήσει σωστά, ο χρήστης βεβαιώνεται για την ομαλή λειτουργία του server και της σχετικής υπηρεσίας.

Η ιδέα πίσω από το heartbeat είναι αρκετά απλή. Μόνο που στην πράξη η υλοποίησή της έκρυβε ένα τεράστιο σφάλμα! Βλέπετε, ο κώδικας του OpenSSL δεν πραγματοποιούσε κανέναν έλεγχο για το μέγεθος της απάντησης που ζητούσε ο χρήστης. Έτσι, αν ο χρήστης έστελνε ένα μήνυμα των 16 bytes και ζητούσε σαν απάντηση 128 bytes, ο server θα ικανοποιούσε το σχετικό αίτημα χωρίς το παραμικρό παράπονο. Αναρωτιέστε που θα έβρισκε τα 112 πρόσθετα bytes; Εν ολίγοις, θα συνέβαινε αυτό που ονομάζουμε buffer overrun. Ο κώδικας θα έπαιρνε ολόκληρο το μήνυμα από την προσωρινή θέση αποθήκευσης στη μνήμη (buffer) και θα συνέχιζε την ανάγνωση υπερβαίνοντας τα όρια αυτής της περιοχής (overrun), μέχρι να συμπληρώσει τα 128 bytes. Τελικά, ο χρήστης θα λάμβανε όλο το αρχικό μήνυμα *μαζί* με 112 bytes από τη μνήμη του server. Σημειώστε τώρα ότι στον κώδικα του OpenSSL, η μεταβλητή που διατηρεί το μέγεθος της απάντησης είναι τύπου unsigned int κι έχει μήκος 16bit. Αυτό σημαίνει ότι το μέγεθος της απάντησης που ζητάει ο χρήστης μπορεί να φτάσει έως και τα 64KB (2^16 bytes). Επομένως, με μία μόνο εκτέλεση της παραπάνω απάτης, ένας επιτιθέμενος θα μπορούσε να διαβάσει ένα σημαντικά μεγάλο κομμάτι από τη μνήμη του server.

Όπως καταλαβαίνετε, το πρόβλημα εντοπιζόταν στην "αφέλεια" που επιδείκνυε το OpenSSL, απέναντι στα αιτήματα του χρήστη. Ο προγραμματιστής που ενσωμάτωσε τον μηχανισμό heartbeat στο OpenSSL είχε ξεχάσει να κάνει αυτό που ονομάζουμε data validation. Κι όπως γνωρίζετε πολύ καλά, π.χ., από όλα τα άρθρα του περιοδικού που καταπιάνονται με την PHP και την ανάπτυξη Διαδικτυακών εφαρμογών, αυτή η παράλειψη έχει καταστροφικές συνέπειες.

Βουτιά στον κώδικα

Ο κώδικας του OpenSSL είναι γραμμένος σε C. Ωστόσο, ακόμα κι αν δεν έχετε ασχοληθεί ποτέ με τη συγκεκριμένη γλώσσα, είμαστε σίγουροι ότι δεν θα δυσκολευτείτε καθόλου να κατανοήσετε το πρόβλημα. Παρακάτω φαίνεται ένα απόσπασμα της ρουτίνας που εξυπηρετεί τα αιτήματα heartbeat. Αν θέλετε να δείτε ολόκληρο τον κώδικα της ρουτίνας, μπορείτε να κατεβάσετε τον πηγαίο κώδικα κάποιας από τις ευπαθείς εκδόσεις του OpenSSL και να ανοίξετε το αρχείο /ssl/d1_both.c. Παρεμπιπτόντως, αν κι όσοι διατηρούν τον δικό τους server θα το έχουν ήδη υπόψη, να αναφέρουμε ότι οι ευπαθείς εκδόσεις του προγράμματος είναι οι 1.0.1 έως και 1.0.1f.

```
int dtls1_process_heartbeat(SSL *s)
  {
  unsigned char *p = &s->s3->rrec.data[0], *pl;
  unsigned short hbtype;
  unsigned int payload;
  unsigned int padding = 16; /* Use minimum padding */
  /* Read type and payload length first */
  hbtype = *p++;
  n2s(p, payload);
  pl = p;
[...]
  if (hbtype == TLS1_HB_REQUEST)
    {
    unsigned char *buffer, *bp;
    int r;
    /* Allocate memory for the response, size is 1 byte
     * message type, plus 2 bytes payload length, plus
     * payload, plus padding
     */
    buffer = OPENSSL_malloc(1 + 2 + payload + padding);
    bp = buffer;
    /* Enter response type, length and copy payload */
    *bp++ = TLS1_HB_RESPONSE;
    s2n(payload, bp);
    memcpy(bp, pl, payload);
    bp += payload;
```

Τα κρίσιμα σημεία του κώδικα, εκεί όπου εντοπίζεται κι εκδηλώνεται η ευπάθεια, είναι οι μαρκαρισμένες γραμμές. Ας δούμε όμως τι ακριβώς συμβαίνει στον κώδικα. Η ρουτίνα ξεκινά με τη δήλωση ενός δείκτη (p). Ο συγκεκριμένος δείκτης παραπέμπει προς τη θέση μνήμης όπου βρίσκεται το αίτημα του χρήστη. Τα σχετικά δεδομένα βέβαια δεν έχουν αποθηκευτεί "χύμα" και με τυχαίο τρόπο στη μνήμη, αλλά κατοικούν σε μία δομή δεδομένων. Πρόκειται για μια δομή δεδομένων που ορίζεται στον κώδικα του OpenSSL και η οποία περιλαμβάνει τον τύπο του εισερχόμενου αιτήματος, το μέγεθος της απάντησης που ζητά ο χρήστης, το μήνυμα που ενσωμάτωσε στο αίτημά του κ.ά. Αν δεν σας τρομάζουν όλα αυτά, ρίξτε μια ματιά και στον ορισμό της σχετικής δομής δεδομένων:

```
typedef struct ssl3_record_st
  {
  int type;              /* type of record */
  unsigned int length;   /* How many bytes available */
  unsigned int off;      /* read/write offset into 'buf' */
  unsigned char *data;   /* pointer to the record data */
  unsigned char *input;  /* where the decode bytes are */
  unsigned char *comp;   /* only used with decompression - malloc()
ed */
  unsigned long epoch;   /* epoch number, needed by DTLS1 */
  unsigned char seq_num[8]; /* sequence number, needed by DTLS1 */
  } SSL3_RECORD;
```

Αν όλα τα παραπάνω σας φαίνονται ακατανόητα, δεν υπάρχει λόγος ανησυχίας. Αρκεί να συγκρατήσετε ότι όλα τα δεδομένα που έστειλε ο χρήστης βρίσκονται αποθηκευμένα κάπου στη μνήμη. Αυτή η θέση μνήμης διατηρείται στο δείκτη p.

Ο κώδικας της ρουτίνας του Heartbeat συνεχίζει με τη δήλωση μερικών ακόμα μεταβλητών, για τον τύπο του αιτήματος, το μέγεθος της απάντησης που ζητά ο χρήστης κ.λπ. Μέχρι εδώ είναι όλα καλά. Το έγκλημα ξεκινά λίγο παρακάτω! Ο κώδικας χρησιμοποιεί τον δείκτη p και διαβάζει το πρώτο byte από την δομή δεδομένων που περιλαμβάνει το αίτημα του χρήστη. Η τιμή αυτού του byte δηλώνει τον τύπο του αιτήματος κι αποθηκεύεται στη μεταβλητή hbtype. Ούτε κι αυτό είναι κακό, αφού ο τύπος του αιτήματος είναι ακίνδυνος — όσο κι αν αλλοιωθεί. Στη συνέχεια, όμως, ο κώδικας χρησιμοποιεί και πάλι τον δείκτη p, για να διαβάσει τα επόμενα δύο bytes του εισερχόμενου αιτήματος. Οι τιμές αυτών των bytes αποθηκεύονται σαν ένας αριθμός των 16bit, στη μεταβλητή payload. Αυτή η ανάγνωση και η μετατροπή πραγματοποιείται με τη βοήθεια μιας εντολής macro (n2s) που έχει ορίσει ο προγραμματιστής. Παρακάτω φαίνεται ο ορισμός της συγκεκριμένης εντολής:

```
#define n2s(c,s) ((s=(((unsigned int)(c[0]))<< 8) | \
                 (((unsigned int)(c[1])) )), c+=2)
```

Αν οι παραπάνω γραμμές μοιάζουν να στερούνται κάθε νοήματος, μπορείτε και πάλι να τις αγνοήσετε. Τις παραθέσαμε μόνο για τους περίεργους :) Το σημαντικό εδώ, είναι αυτό που αναφέραμε προηγουμένως: Ο κώδικας διαβάζει δύο bytes από το εισερχόμενο αίτημα και τα αποδίδει, ως μία τιμή των 16bit, στη μεταβλητή payload. Πρόκειται για ένα απόλυτα λογικό βήμα στην "εξέλιξη" της ρουτίνας. Η μεταβλητή payload προορίζεται για την αποθήκευση του μεγέθους της απάντησης που ζητά

ο χρήστης κι αυτό το μέγεθος βρίσκεται όντως στα συγκεκριμένα δύο bytes του αιτήματος. Το παράλογο της υπόθεσης έγκειται στην ολοκληρωτική απουσία οποιουδήποτε ελέγχου, πριν ή μετά την ανάθεση της τιμής. Έτσι, ο χρήστης αποδίδει αυθαίρετα οποιαδήποτε τιμή θέλει σε μια κρίσιμη μεταβλητή του προγράμματος! Από 'κεί κι έπειτα, η καταστροφή είναι σχεδόν αναπόφευκτη…

Λίγο παρακάτω, ο κώδικας ελέγχει αν το εισερχόμενο αίτημα είναι τύπου heartbeat (TLS1_HB_REQUEST). Εφόσον ισχύει κάτι τέτοιο, ξεκινά η προετοιμασία της απάντησης. Το πρόγραμμα δημιουργεί έναν προσωρινό αποθηκευτικό χώρο στη μνήμη, για την απάντηση που θα σταλεί στο χρήστη. Σε αυτή την περιοχή, μεταξύ άλλων αποθηκεύονται όσα bytes ορίζει η τιμή της μεταβλητής payload. Η αντιγραφή πραγματοποιείται με την εντολή memcpy και τα δεδομένα προέρχονται από την περιοχή της μνήμης στην οποία παραπέμπει ο δείκτης pl. Το πού ακριβώς παραπέμπει ο εν λόγω δείκτης είναι φανερό από τον κώδικα της ρουτίνας, αλλά δεν έχει καμία σημασία για την περιγραφή μας. Η ζημιά έχει γίνει! Η απάντηση που θα λάβει ο χρήστης, θα περιλαμβάνει όσα bytes ζήτησε. Αν ο χρήστης υπήρξε πονηρός κι έστειλε ένα πολύ μικρότερο μήνυμα σε σχέση με εκείνο που ζήτησε από τον server, θα καταφέρει να διαβάσει μια περιοχή της μνήμης στην οποία, υπό φυσιολογικές συνθήκες, δεν θα είχε καθόλου πρόσβαση.

Αντιμετώπιση

Οι δομές δεδομένων, οι δείκτες και γενικότερα το στρυφνό και λακωνικό συντακτικό της C προκαλούν σύγχυση σε όλους όσοι έρχονται αντιμέτωποι μαζί τους για πρώτη φορά. Ωστόσο, θέλουμε να πιστεύουμε ότι κατανοήσατε πλήρως το αδύναμο σημείο στον κώδικα του OpenSSL. Στο κάτω-κάτω, πρόκειται για μία και μόνο παράλειψη: Την απουσία κάθε ελέγχου των δεδομένων που ορίζει ο χρήστης. Ακριβώς γι' αυτό, η αντιμετώπιση της ευπάθειας επιτεύχθηκε με ελάχιστες προσθήκες. Παρακάτω φαίνεται η πλέον καθοριστική επέμβαση που πραγματοποιήθηκε στον κώδικα:

```
/* Read type and payload length first */
if (1 + 2 + 16 > s->s3->rrec.length)
    return 0;
hbtype = *p++;
n2s(p, payload);
if (1 + 2 + payload + 16 > s->s3->rrec.length)
    return 0;
```

Όπως βλέπετε, οι γραμμές που προστέθηκαν (οι μαρκαρισμένες) βρίσκονται στην αρχή της ρουτίνας και γύρω από τον ορισμό της μεταβλητής payload. Ουσιαστικά, πρόκειται για δύο απλούς ελέγχους. Ο πρώτος αφορά στο μέγεθος του μηνύματος που στέλνει ο χρήστης, μέρος του οποίου θα πρέπει να ενσωματωθεί στην απάντηση του server. Αν αυτό το μήνυμα είναι κενό, τότε το συνολικό μέγεθος του αιτήματος του χρήστη θα είναι μικρότερο από το ελάχιστο μέγεθος των αιτημάτων (1 byte για το είδος του αιτήματος συν 2 bytes για το μέγεθος της απάντησης συν 16 συμπλη-

ρωματικά bytes = 19 bytes). Κι όπως αντιλαμβάνεστε, ένας χρήστης που στέλνει κενό μήνυμα, ακόμα κι αν ζητήσει ένα μόλις byte σαν απάντηση, είναι σίγουρο ότι προσπαθεί να ξεγελάσει τον server και να διαβάσει ύπουλα τη μνήμη. Έτσι, αν ο έλεγχος είναι επιτυχημένος, η ρουτίνα τερματίζεται (return) χωρίς να σταλεί καμία απάντηση στο χρήστη. Ο δεύτερος έλεγχος συγκρίνει το μέγεθος της απάντησης που ζητά ο χρήστης, με εκείνο του εισερχόμενου αιτήματος. Αν το πρώτο είναι μεγαλύτερο από το δεύτερο (αν ζητά περισσότερα από όσα έστειλε), είναι φανερό ότι προσπαθεί και πάλι να ξεγελάσει το server. Αν όχι, η εκτέλεση της ρουτίνας συνεχίζεται κανονικά, αφού η απάντηση θα περιλαμβάνει μόνο ένα τμήμα του αρχικού μηνύματος του χρήστη και τίποτα περισσότερο.

Αν θέλετε να δείτε όλες τις αλλαγές που πραγματοποιήθηκαν στον πηγαίο κώδικα του OpenSSL για την αντιμετώπιση του Heartbleed, αρκεί να επισκεφθείτε την ακόλουθη σελίδα:

`http://bit.ly/openssl-commitdiff`

Εμείς θα εκμεταλλευτούμε την ευκαιρία για λίγο κήρυγμα και θα τονίσουμε (ξανά) την τεράστια σημασία του data validation. Όπως είδατε, η παράλειψη αυτού του μέτρου μπορεί να έχει απρόβλεπτες και εξαιρετικά δυσάρεστες συνέπειες. Κι επειδή δεν μπορούμε να κρατηθούμε άλλο, θα σας αφήσουμε με μερικά ερωτήματα που βασανίζουν κι εμάς τους ίδιους: Πώς γίνεται να "επέζησε" ένα τέτοιο bug στον πηγαίο κώδικα μιας τόσο σημαντικής εφαρμογής; Με την ανάπτυξη του OpenSSL ασχολούνται περί τους δέκα προγραμματιστές. Είναι δυνατόν να ασχολούνται τόσο λίγοι άνθρωποι με ένα τόσο κρίσιμο πρόγραμμα; Κι εξάλλου, γιατί να ασχολούνται εθελοντικά και στον ελεύθερό τους χρόνο, αντί να εργάζονται κανονικά; Και για να \`χουμε καλό ερώτημα: Η NSA, δεν ήξερε τίποτα;

Ίσως έχετε να πείτε κάτι γι' αυτά τα ερωτήματα, ίσως και όχι. Σε κάθε περίπτωση, προτείνουμε να σπεύσετε και ν' αλλάξετε τα passwords σας από όλες τις κρίσιμες διαδικτυακές υπηρεσίες που χρησιμοποιείτε. Αν μάλιστα έχετε το LastPass, να ξέρετε ότι περιλαμβάνει έναν μηχανισμό ο οποίος, με βάση τα sites που επισκέπτεστε, διεξάγει μια σειρά ελέγχων και σας λέει σε ποια πρέπει οπωσδήποτε να ορίσετε νέο password. Πηγαίνετε στην επιλογή "Tools" του LastPass, διαλέξτε το "Security Test" και ξεκινήστε το.

Skill: Intermediate
Tags: LocalStorage, SessionStorage, cookies, JSON, JavaScript



Το νέο, βολικό WebStorage και μια ύπουλη αδυναμία

Η πιο πολυσυζητημένη, γλυκιά συντροφιά στο web, τα cookies, έχουν αρχίσει να δίνουν σιγά σιγά τη θέση τους σε ένα νέο, μοντέρνο τρόπο αποθήκευσης δεδομένων στην πλευρά του web browser. Στο παρόν άρθρο θα μελετήσουμε το WebStorage API δημιουργώντας τη δική μας offline web εφαρμογή, ενώ δεν θα παραλείψουμε να εξετάσουμε τη νέα αυτή τεχνολογία κι από τη σκοπιά των κενών ασφαλείας.

by multiPetros

Το 1996, τότε που ο παγκόσμιος ιστός έκανε τα πρώτα του βήματα, μια τεχνολογική πρόταση της Netscape ήρθε για να μείνει. Η υλοποίηση της εν λόγω πρότασης, που παρά την καθολική της επικράτηση ποτέ δεν έγινε επίσημο standard, δεν είναι άλλη από τα γνωστά σε όλους μας cookies. Πρόκειται για αρχεία κειμένου με μέγεθος έως 4KB, τα οποία αποθηκεύονται τοπικά, στους υπολογιστές των επισκεπτών δικτυακών τόπων. Τα cookies περιέχουν ρυθμίσεις, προτιμήσεις κι άλλα στοιχεία που αξιοποιούνται από τα αντίστοιχα web sites, π.χ., για προβολή εξατομικευμένου περιεχομένου, ως προσωρινή μνήμη, για τη "στήριξη" άλλων τεχνικών (π.χ., PHP sessions[1]), για την καταγραφή προτιμήσεων κ.ο.κ. Ειδικά γι' αυτό το τελευταίο, την καταγραφή των προτιμήσεων, έχουν ειπωθεί πάρα πολλά και έχουν γίνει απειραριθμήσιμα keystrokes. Παρά την κατά καιρούς δαιμονοποίηση των cookies — και καθώς ο τρόπος χρήσης τους είναι θέμα επιλογών από πλευράς των χρηστών—, δίχως να θέλουμε να υπεισέλθουμε σε συζητήσεις περί ιδιωτικότητας οφείλουμε να παραδεχθούμε ότι ο παγκόσμιος ιστός εξελίχθηκε στη σημερινή του μορφή ως ένα βαθμό χάρη στα cookies.

Φυσικά, από το '96 έχουν περάσει αρκετά χρόνια, τα οποία με τους ρυθμούς ανάπτυξης του παγκόσμιου ιστού φαντάζουν ως αιώνες. Τα cookies, αν και συνεχίζουν να υπάρχουν και να χρησιμοποιούνται, έχουν αρχίσει να δείχνουν τα χρόνια τους. Οι ανάγκες για τοπική αποθήκευση δεδομένων ιστοσελίδων εξελίχθηκαν και αυξήθηκαν. Ειδικά δε με την επανάσταση των πανταχού παρόντων, πλέον, φορητών συσκευών, η ανάγκη για τοπική αποθήκευση έχει γιγαντωθεί.

Storage API

Το Storage API είναι η εξέλιξη των cookies. Μπορεί να χρησιμοποιηθεί από τους δικτυακούς τόπους ή τις web εφαρμογές αποκλειστικά — ή σε συνδυασμό με τα cookies. Το Storage API υποστηρίζει δύο είδη τοπικής αποθήκευσης: Το Session-Storage, που αφορά στην προσωρινή αποθήκευση δεδομένων για όση ώρα έχετε ανοιχτό τον web browser, καθώς και το LocalStorage, που επιτρέπει τη μονιμότερη αποθήκευση δεδομένων και είναι κι αυτό που συνήθως μας αφορά κατά την ανάπτυξη web εφαρμογών. Από 'δώ και στο εξής θα εξετάσουμε και θα αναφερόμαστε σ' αυτό το δεύτερο, το LocalStorage. Ας γνωρίζετε πάντως πως, ό,τι κι αν πούμε γι' αυτό, πέραν της διαφοράς στη διάρκεια ζωής των αποθηκευμένων δεδομένων, ισχύει και για το SessionStorage.

Με το μηχανισμό LocalStorage κάθε δικτυακός τόπος έχει τη δυνατότητα να αποθηκεύει στο μηχάνημα του χρήστη τουλάχιστον 5MB, ένα νούμερο το οποίο μπορεί να είναι και μεγαλύτερο — ανάλογα με την υλοποίηση του εκάστοτε browser ή/και των ρυθμίσεων του χρήστη. Στην περίπτωση που το άνω όριο ξεπεραστεί, επιστρέφεται στον κώδικα της JavaScript ένα σφάλμα QUOTA_EXCEEDED_ERR.

Και μιας και μιλήσαμε για JavaScript, παρακαλούμε αποβάλλετε την όποια ταχυκαρδία σας προκάλεσε η αναφορά αυτή. Ο κώδικας των παραδειγμάτων που θα εξετάσουμε στη συνέχεια θα είναι γραμμένος κατά κύριο λόγο σε JavaScript και HTML, όμως δεν θα επιχειρήσουμε να μάθουμε από την αρχή την πρώτη. Ευτυχώς το συντακτικό της, ως C-like γλώσσα, δεν διαφέρει και πολύ από αυτό της PHP, την οποία εξετάσαμε στα οκτώ άρθρα της σειράς μας περί ανάπτυξης εφαρμογών LAMP.

Αναφορικά τώρα με την αποθήκευση των δεδομένων στο LocalStorage, αυτή γίνεται

1. Εάν η αυτό το "PHP session" δεν σας λέει κάτι διαβάστε το τελευταίο μέρος της σειράς "Ανάπτυξη web εφαρμογών σε περιβάλλον LAMP", στο τεύχος 030 του deltaHacker.

στη βάση της λογικής των ζευγαριών κλειδιού-τιμής. Έτσι, χρησιμοποιώντας το όνομα του κλειδιού μπορούμε με εύκολο τρόπο να προσπελάσουμε τα δεδομένα που έχουμε αποθηκεύσει σ' αυτό. Εκτός όμως από κείμενο ή τα δεδομένα μιας μεταβλητής, το LocaleStorage μπορεί να φιλοξενήσει και αντικείμενα. Πολύτιμος σύμμαχός μας σ' αυτό θα είναι το JSON API.

Για την αποθήκευση δεδομένων στη μνήμη του LocalStorage, το Storage API παρέχει τη συνάρτηση **setItem(name, value)**, όπου name είναι το όνομα με το οποίο θα έχουμε πρόσβαση στην τιμή που περιέχει το value.

Για την άντληση των δεδομένων μπορούμε να χρησιμοποιήσουμε τη συνάρτηση **getItem(name)**, η οποία μας επιστρέφει την τιμή που αντιστοιχεί στο ζεύγος στο οποίο αναφέρεται το name.

Η δε διαγραφή ενός ζεύγους γίνεται με χρήση της **removeItem(name)**. Για τη διαγραφή όλων των αποθηκευμένων δεδομένων του LocalStorage υπάρχει η **clear()**.

Επίσης χρήσιμη είναι και η ιδιότητα **length**, η οποία μας επιστρέφει το πλήθος των στοιχείων που έχουμε αποθηκεύσει στο LocalStorage. Αυτό σε συνδυασμό με τη συνάρτηση **key(position)**, που επιστρέφει το όνομα του στοιχείου για την εκάστοτε θέση που περνάμε στην **key**, μας επιτρέπει να ανατρέξουμε όλη τη συλλογή των στοιχείων που βρίσκονται αποθηκευμένα στο LocalStorage.

JSON

Η λέξη JSON (προφέρεται "τζέισον") προέρχεται από τα αρχικά των λέξεων JavaScript Object Notation κι αποτελεί ένα ανοιχτό πρότυπο format που χρησιμοποιεί κείμενο, εύκολα αναγνώσιμο κι από ανθρώπους, για τη μετάδοση ή την αποθήκευση αντικειμένων. Τα δεδομένα κι εδώ απεικονίζονται από ζεύγη χαρακτηριστικών-τιμών. Το JSON είναι ιδιαιτέρως αγαπητό στους απανταχού developers, οι οποίοι το προτιμούν για το λεγόμενο "object serialization". (Από την άλλη, υπάρχει και το δύστροπο κι απαιτητικό σε χρόνο επεξεργασίας και σε όγκο δεδομένων SOAP, το οποίο βασίζεται στην XML.) Και μη σας μπερδεύει η λέξη JavaScript: το JSON είναι τελείως ανεξάρτητο από γλώσσες προγραμματισμού.

Αν και δεν θα μας απασχολήσουν "χειροκίνητα" δεδομένα, serialized κατά JSON, καλό είναι να γνωρίζουμε το πώς αναπαρίστανται τα εν λόγω δεδομένα.

Στο συντακτικό του JSON, η αναπαράσταση ενός αντικειμένου ξεκινάει με "{" (αριστερό άγκιστρο) και τελειώνει με "}" (δεξιό άγκιστρο). Κάθε όνομα-μέλος ακολουθείται από ":" (άνω-κάτω τελεία) και τα ζευγάρια μέλους/τιμής χωρίζονται με "," (κόμμα).

Η αναπαράσταση ενός πίνακα ξεκινάει με "[" (αριστερή αγκύλη) και τελειώνει με "]" (δεξιά αγκύλη). Οι τιμές των θέσεων του πίνακα μπαίνουν εντός των αγκυλών και μεταξύ τους χωρίζονται με "," (κόμμα).

Μία τιμή μπορεί να είναι string, το οποίο μπαίνει ανάμεσα σε " (διπλά εισαγωγικά) ή αριθμός, true, false, null. Επιπλέον, μια τιμή μπορεί να είναι αντικείμενο ή πίνακας — κι αναπαρίσταται όπως είδαμε προηγουμένως. Τα strings μπορούν να περιέχουν χαρακτήρες Unicode, ενώ για την αναπαράσταση ειδικών χαρακτήρων (π.χ., εκείνον της αλλαγής γραμμής) επιστρατεύεται ο χαρακτήρας διαφυγής "\" (η ανάποδη κάθετος).

Το JSON API παρέχει δύο πολύ χρήσιμες συναρτήσεις, τις οποίες σκοπεύουμε να αξιοποιήσουμε για να επιτύχουμε τους σκοπούς μας. Από αυτές η πρώτη είναι η **stringify(variable)**, η οποία επιτυγχάνει serialization (κατά τις προδιαγραφές του JSON) της μεταβλητής που της περνάμε ως παράμετρο. Μας επιστρέφει δηλαδή μια συμβολοσειρά, η οποία αναπαριστά τα δεδομένα μεταβλητής — πάντα κατά τις προδιαγραφές του JSON.

Η δεύτερη συνάρτηση είναι η **parse(jsonString)**, στην οποία περνάμε μια έγκυρη συμβολοσειρά JSON και μας επιστρέφει τα δεδομένα της serialized μεταβλητής, είτε πρόκειται για κάποιο scalar τύπο (αριθμό, συμβολοσειρά, λογική τιμή) είτε για πίνακα ή αντικείμενο.

Ένα web app στον browser μας

Όπως έχετε καταλάβει, εμείς, εδώ στην Parabing Creations, είμαστε πρακτικοί άνθρωποι και δεν μας αρέσει να μένουμε στη θεωρία. Έτσι, για τους σκοπούς του άρθρου θα δημιουργήσουμε βήμα προς βήμα μια web εφαρμογή "to-do list", η οποία παρά τη web-οστρεφή της φύση δεν θα βασίζεται ούτε θα επικοινωνεί με κάποιον web server. Αντιθέτως, θα ζει και θ' αναπνέει μέσα στον αγαπημένο μας web browser, αξιοποιώντας το Storage API. Οι αποθηκευμένες σημειώσεις θα είναι διαθέσιμες κάθε φορά που ανοίγεται στον browser η αντίστοιχη σελίδα — μέχρι φυσικά να τις διαγράψετε.

Ξεκινάμε με την παράθεση του κώδικα κι εν συνεχεία τον αναλύουμε γραμμή προς γραμμή. Ο κώδικας της εφαρμογής είναι διαθέσιμος κι από το `http://bit.ly/dh031todo`.

```
<!doctype html>
<html lang="el">
<head>
    <meta charset="utf-8">
    <title>LocalStorage to-do List</title>
    <style type="text/css">
        label.done{
            text-decoration:line-through ;
            color: #b3b3b3;
        }
        #todo-list{
            padding: 10px 10px 25px 5px ;
        }
    </style>
    <script>
        const TODOS = 'todos' ;

```

```
function updateTodo(){
    var savedList = new Array() ;
    var currentList = document.getElementById('todo-list') ;
    var itemsCode = '' ;
    if(localStorage.getItem(TODOS) != null){
        savedList = JSON.parse(localStorage.getItem(TODOS)) ;
    }
    currentList.innerHTML = '' ;
    for(var i=0; i<savedList.length; i++){
        var currenLabel = document.createElement('label') ;
        var currentInput = document.createElement('input') ;
        currenLabel.setAttribute('name', 'todo-item') ;
        currenLabel.setAttribute('id', 'todo-item-' + i) ;
        currentInput.setAttribute('type', 'checkbox') ;
        currentInput.setAttribute('name', 'todo-checkbox') ;
        if(savedList[i].done == true){
            currentInput.setAttribute('checked','checked') ;
            currenLabel.setAttribute('class','done') ;
        }
        var todoText = document.createTextNode(savedList[i].txt) ;

        currenLabel.appendChild(currentInput) ;
        currenLabel.appendChild(todoText) ;
        currentList.appendChild(currenLabel) ;
        currentList.appendChild(document.createElement('br')) ;
    }
}

function addTodo(){
    var userInput = document.getElementById('todo-text').value ;
    if(userInput.length > 0){
        var savedList = new Array() ;
        var todoObj = new Object() ;
        todoObj.txt = userInput ;
        todoObj.done = false ;
        if(localStorage.getItem(TODOS) != null){
```

```
                savedList = JSON.parse(localStorage.getItem(TODOS)) ;
            }
            savedList.push(todoObj) ;
            localStorage.setItem(TODOS, JSON.stringify(savedList)) ;
            updateTodo() ;
        }else{
            alert('Please enter some text!\n;-)') ;
        }
    }

    function removeTodo(){
        var savedList = new Array() ;
        var newList = new Array() ;
        if(localStorage.getItem(TODOS) != null){
            savedList = JSON.parse(localStorage.getItem(TODOS)) ;
        }
        checkboxes = document.getElementsByName('todo-checkbox') ;
        for(var i=0; i<checkboxes.length; i++){
            if(!checkboxes[i].checked){
                newList.push(savedList[i]) ;
            }
        }
        localStorage.setItem(TODOS, JSON.stringify(newList)) ;
        updateTodo() ;
    }

    function clearTodo(){
        if(localStorage.getItem(TODOS) != null){
            if(confirm('Delete permanently all TO-DOs ?')){
                localStorage.removeItem(TODOS) ;
                updateTodo() ;
            }
        }
    }

    function doneTodo(){
```

```
090             var savedList = new Array() ;
091             if(localStorage.getItem(TODOS) != null){
092                 savedList = JSON.parse(localStorage.getItem(TODOS)) ;
093             }
094             checkboxes = document.getElementsByName('todo-checkbox') ;
095             for(var i=0; i<checkboxes.length; i++){
096                 if(checkboxes[i].checked){
097                     savedList[i].done = true ;
098                 }else{
099                     savedList[i].done = false ;
100                 }
101             }
102             localStorage.setItem(TODOS, JSON.stringify(savedList)) ;
103             updateTodo() ;
104         }
105     </script>
106 </head>
107 <body onload="updateTodo()">
108     <h1>ToDo List</h1>
109     <p>Add new todo:</p>
110     <input type="text" id="todo-text">
111     <input type="button" value="Store" onclick="addTodo()">
112     <div id="todo-list"></div>
113     <input type="button" value="Mark Selected as Done" onclick="doneTodo()"><br>
114     <input type="button" value="Remove Selected" onclick="removeTodo()"><br>
115     <input type="button" value="Clear List" onclick="clearTodo()"><br>
116 </body>
117 </html>
```

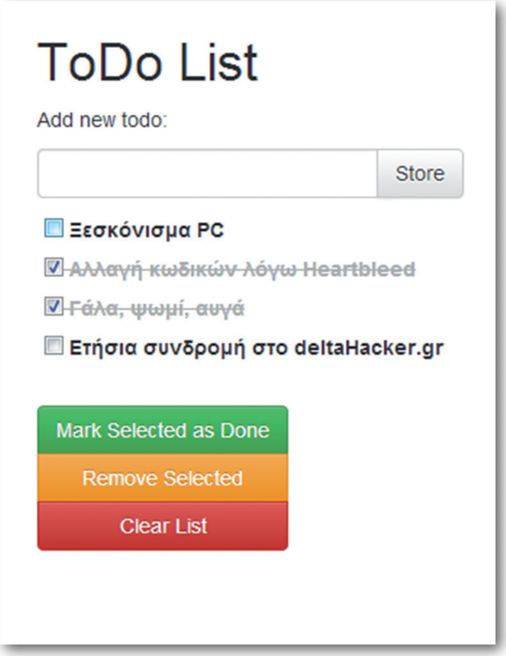
Ξεκινάμε την ανάλυσή μας λίγο ανάποδα, κοιτάζοντας πρώτα στον HTML κώδικα που βρίσκεται στο τέλος του αρχείου. Εκεί, στη γραμμή 107, όπου ξεκινάει η ετικέτα body, ορίζουμε ότι κατά την φόρτωση της σελίδας (συμβάν onload), θα εκτελείται η JavaScript συνάρτηση updateTodo(). Η συγκεκριμένη συνάρτηση είναι υπεύθυνη για την αλίευση των αποθηκευμένων δεδομένων και τη φόρτωσή τους σε εμφανίσιμη μορφή στη σελίδα μας.

Συνεχίζοντας, στη γραμμή 110, αντικρίζουμε μια ετικέτα input, η οποία δημιουργεί ένα textbox, στο οποίο θα εισάγουμε το κείμενο της σημείωσής μας. Αμέσως μετά, στη γραμμή 109, υπάρχει πάλι ένα input tag, το οποίο εμφανίζει ένα κουμπί

καταχώρησης της σημείωσης, καλώντας τη συνάρτηση addTodo() όταν ο χρήστης κάνει κλικ σ' αυτό (συμβάν onclick).

Έπειτα, στη γραμμή 112, υπάρχει μια ετικέτα div, με id todo-list. Το όνομά της μας προϊδεάζει για τη χρησιμότητά της, η οποία δεν είναι άλλη από την εισαγωγή και φιλοξενία, εντός της ετικέτας αυτής, των σημειώσεων που θα εισάγουμε.

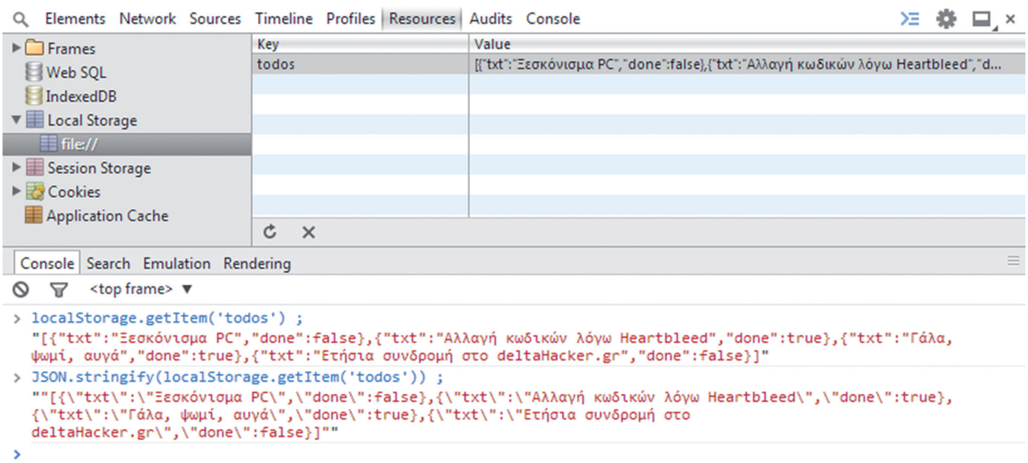
Και, τέλος, στις γραμμές 113 έως και 115 υπάρχουν τρεις ετικέτες input, οι οποίες δημιουργούν τρία κουμπιά. Με το πάτημα του πρώτου κουμπιού, καλείται η συνάρτηση doneTodo(), η οποία μαρκάρει τις επιλεγμένες σημειώσεις ως ολοκληρωμένες. Με το δεύτερο κουμπί καλείται η συνάρτηση ramoveTodo(), η οποία διαγράφει τις επιλεγμένες σημειώσεις. Με το τρίτο κουμπί καλείται η συνάρτηση clearTodo(), η οποία διαγράφει όλη τη λίστα των σημειώσεων.



Η εφαρμογή μας εν δράσει! Φυσικά, δεν θα μπορούσαμε να μη βελτιώσουμε λίγο την εμφάνισή της. Γι' αυτό κι εμπλουτίσαμε τον αρχικό μας κώδικα με τις βιβλιοθήκες του Bootstrap Framework. Στο ZIP αρχείο που σας δίνουμε θα βρείτε και τις δύο εκδόσεις.

Κι εδώ τελειώνει ο HTML κώδικάς μας. Απλοϊκά, αλλά πλήρως λειτουργικά. Ας πάμε τώρα στον JavaScript κώδικα της σελίδας μας, ο οποίος ξεκινάει από τη γραμμή 016. Εκεί έχουμε ορίσει την σταθερά TODOS, η οποία θα περιέχει το κλειδί στο οποίο θα αποθηκεύουμε τα δεδομένα μας στο LocalStorage.

Ξεκινώντας με τη συνάρτηση updateTodo(), στη γραμμή 018, δημιουργούμε έναν πίνακα με όνομα savedList, στον οποίο θα φορτώσουμε τις αποθηκευμένες σημειώσεις. Φτιάχνουμε επίσης και μια μεταβλητή με όνομα currentList, μέσω της οποίας θα έχουμε πρόσβαση στις ετικέτες που βρίσκονται εντός του div με id todo-list (ουσιαστικά, στις ετικέτες που αναπαριστούν τις σημειώσεις μας). Έπειτα, στη γραμμή 022, ελέγχουμε αν υπάρχει αποθηκευμένη λίστα στη LocalSorage. Αν ναι, η συνάρτηση αναδημιουργεί τον πίνακα με τη βοήθεια της JSON.parse(element), και τον φορτώνει στη μεταβλητή savedList. Έπειτα, στη γραμμή 025, εκχωρούμε μια κενή συμβολοσειρά στην ιδιότητα της innerHTML, της μεταβλητής currentList, διαγράφοντας ουσιαστικά οτιδήποτε υπάρχει εντός του div todo-list. Έπειτα, ανατρέχοντας ένα προς ένα τα στοιχεία του πίνακα savedList, δημιουργούμε με τη συνάρτηση document.createElement τις κατάλληλες ετικέτες label και input (check boxes), οι οποίες παρουσιάζουν τα αποθηκευμένα στοιχεία μας. Παρατηρείστε ότι στην περίπτωση που το αντικείμενο που εξετάζουμε έχει την τιμή true για το μέλος done, ορίζουμε για το συγκεκριμένο label την ιδιότητα class σε done, έτσι ώστε το κείμενο να εμφανιστεί με γκρίζο χρώμα και διακριτή διαγραφή, όπως ορίζεται στις οδηγίες του ενσωματωμένου stylesheet (γραμμές 007-013). Αφού ολοκληρώσουμε τη δημιουργία των ετικετών και τον ορισμό των ιδιοτήτων τους (συναρτήσεις setAttribute), τις προσθέτουμε στο κάτω μέρος του δέντρου των ετικετών του div todo-list, με τη βοήθεια της συνάρτησης appendChild.



Τα περιεχόμενα του LocalStorage, όπως εμφανίζονται από την developer console του browser μας. Παρατηρήστε τον τρόπο μετατροπής της αποθηκευμένης τιμής του κλειδιού todos σε JSON string, χρησιμοποιώντας τη μέθοδο stringify.

Στη συνέχεια, στη γραμμή 046, ξεκινάει η συνάρτηση addTodo(), η οποία όπως αναφέραμε προσθέτει μια σημείωση στη λίστα μας. Στη γραμμή 047 ορίζουμε τη μεταβλητή userInput, η οποία παίρνει ως τιμή το περιεχόμενο του textbox, στο οποίο ο χρήστης εισάγει το κείμενο της νέας σημείωσης. Έπειτα, εκτελούμε έναν έλεγχο για το αν ο χρήστης έχει εισάγει κάποιο κείμενο κι αν όχι εμφανίζουμε μια σχετική ειδοποίηση (alert, γραμμή 060). Αλλιώς δημιουργούμε ένα νέο πίνακα με όνομα savedList (γραμμές 049 και 050), στον οποίο θα φορτώσουμε τις ήδη αποθηκευμένες σημειώσεις κι ένα αντικείμενο με όνομα todoObj που θα περιγράφει την κάθε σημείωση και θα έχει δύο μέλη: το txt και το done. Στο πρώτο θα καταχωρείται το κείμενο της σημείωσης και στο δεύτερο μια λογική τιμή, η οποία θα επισημαίνει αν η σημείωση έχει ολοκληρωθεί ή όχι (γραμμές 051 και 051). Ύστερα, στη γραμμή 053 εκτελούμε έναν έλεγχο για την ύπαρξη του κλειδιού todos στη LocalStorage. Αν υπάρχει, τότε με τη βοήθεια της συνάρτησης parse του JSON ανακτούμε τον αποθηκευμένο πίνακα και τον περνάμε στη μεταβλητή savedList. Κατόπιν, γραμμή 056, προσθέτουμε στο τέλος του πίνακα, με χρήση της μεθόδου push(element), το αντικείμενο todoObj που μόλις αρχικοποιήσαμε. Ακολούθως επαναποθηκεύουμε τον πίνακα στη LocalStorage, μετατρέποντάς τον σε ένα JSON string με τη συνδρομή της συνάρτησης stringify(element). Τέλος καλούμε την updateTodo(), έτσι ώστε να επαναφορτωθεί η λίστα των σημειώσεων στη σελίδα μας, περιλαμβάνοντας κι αυτή που μόλις προσθέσαμε.

Σειρά έχει η συνάρτηση removeTodo(), στη γραμμή 064. Η συνάρτηση αυτή φορτώνει, κατά τα γνωστά, τον αποθηκευμένο πίνακα από τη LocalStorage στη μεταβλητή savedList. Κατόπιν φορτώνει έναν πίνακα με όλα τα στοιχεία με όνομα todo-checkbox (ουσιαστικά όλα τα check boxes). Διατρέχοντάς τα ένα προς ένα ελέγχει όσα δεν έχουν τσεκαριστεί και προσθέτει στον πίνακα newList τα αντίστοιχα στοιχεία από τον πίνακα savedList. Δημιουργεί δηλαδή έναν νέο πίνακα, από τον οποίο απουσιάζουν τα επιλεγμένα στοιχεία. Κατόπιν τον αποθηκεύει στη LocalStorage και καλεί την συνάρτηση updateTodo(), για την επανασχεδίαση της σελίδας.



Μεταβαίνοντας στο filldisk.com με τον Firefox (αριστερά), μας εμφανίζεται ένα μήνυμα ότι το demo δεν λειτουργεί στον browser μας. Αντίθετα, με τον Opera (δεξιά) ο τοπικός δίσκος σιγά σιγά αρχίζει να γεμίζει με τα δεδομένα που αποθηκεύουν στο LocalStorage τα άπειρα subdomains της σελίδας. Όσο πιο γρήγορο δίσκο έχουμε, τόσο πιο γρήγορα θα δούμε τη χωρητικότητά του να εξαντλείται...

Ερχόμαστε τώρα στη γραμμή 080 και στη συνάρτηση clearTodo(), η οποία εμφανίζει μια ερώτηση επιβεβαίωσης διαγραφής της λίστας. Αν ο χρήστης απαντήσει θετικά, η συνάρτηση διαγράφει το κλειδί todos από τη LocalStorage (γραμμή 083) και καλεί τη συνάρτηση updateTodo().

Τέλος έχουμε και τη συνάρτηση doneTodo(), η οποία φορτώνει τη λίστα από τη LocalStorage. Επίσης κι όλα τα check boxes και, ελέγχοντας την ιδιότητα checked, περνάει την αντίστοιχη τιμή στη λίστα της LocalStorage (γραμμές 95-101). Εν συνεχεία την αποθηκεύει και επανασχεδιάζει τη σελίδα, καλώντας την updateTodo().

Όπως βλέπετε, με λιγότερες από 120 γραμμές κάναμε το Google Tasks να τρέμει :P

Κυβερνοεπιθέσεις και LocalStorage

Μπορεί να σας φαίνεται παράξενο, το πώς μια τεχνολογία που απλώς χρησιμοποιείται για την τοπική αποθήκευση μερικών megabytes, μπορεί να χρησιμοποιηθεί σε μια επίθεση. Μην πάει ο νους σας σε απόκτηση απομακρυσμένης πρόσβασης στο μηχάνημα του θύματος. Κατ' αρχάς, θα πρέπει να επισημάνουμε ότι δεν είναι η ίδια η τεχνολογία του LocalStorage που μπάζει νερά, αλλά η υλοποίησή του από τους διάφορους κατασκευαστές web browsers. Κι αυτό που μπορεί να επιφέρει μια τέτοια επίθεση είναι το φουλάρισμα του σκληρού δίσκου του υποψήφιου θύματος, με πιθανή συνέπεια την κατάρρευση του browser ή ακόμη και του ίδιου του συστήματος (σε περιπτώσεις που λειτουργεί με περιορισμένους πόρους).

Πώς όμως μπορεί να γεμίσει ο δίσκος του υπολογιστή αφού, όπως αναφέραμε στην αρχή του άρθρου, για κάθε domain υποστηρίζεται ένας περιορισμένος χώρος αποθήκευσης των δεδομένων; (Περί τα 10MB.) Προσέξτε: Τι γίνεται με τα subdomains, είπαμε; Δεν είπαμε! Στους περισσότερους browsers, καθένα απ' αυτά αντιμετωπίζεται ως ξεχωριστό site. Έτσι, σε καθένα απ' αυτά δίνονται άλλα 10MB. Το πρόβλημα λοιπόν εντοπίζεται ακριβώς εδώ. Στον τρόπο, δηλαδή, με τον οποίο οι browsers αντιμετωπίζουν τα subdomains που ζητούν χώρο στη LocalStorage, για την αποθήκευση των δεδομένων τους. Η προδιαγραφή του LocalStorage ορίζει ξεκάθαρα ότι οι browsers θα πρέπει να ορίσουν ένα όριο γι' αυτά τα "φίλια sites", όπως αναφέρονται. Οι κατασκευαστές όμως των browsers, **εκτός του Firefox**, έχουν επιδεικτικά αγνοήσει τη συγκεκριμένη οδηγία, με ό,τι αυτό μπορεί να συνεπάγεται. Να επισημάνουμε ότι στην περίπτωση του Firefox οι μηχανικοί του επέλεξαν τη λύση της κοινής χρήσης του διαθέσιμου χώρου της μνήμης του LocalStorage, από το κύριο domain καθώς κι όλα τα subdomains του.

Γι' αυτό το πρόβλημα έχει ενδιαφερθεί ιδιαίτερα ο Feross Aboukhadijeh, ο οποίος έχει κατασκευάσει κι ένα σχετικό site το οποίο μπορείτε να επισκεφτείτε στο `http://www.filldisk.com`. Από εκεί υλοποιείται μια τέτοια επίθεση (έχετε βεβαίως δυνατότητα να τη διακόψετε), έτσι μπορεί να δει κανείς ζωντανά αν ο αγαπημένος του browser πάσχει από την εν λόγω αδυναμία και συνεπώς είναι ευάλωτος ως προς μια σχετική επίθεση. (Αξίζει δε ιδιαίτερα να μελετήσει κανείς την απόκριση των κατασκευαστών, σχετικά με αυτό το θέμα.)

Το WebStorage, παρά τις όποιες αρχικά επιπόλαιες αποφάσεις από πλευράς κατασκευαστών browsers, φαίνεται ότι ήρθε για να μείνει. Ο εύκολος τρόπος διαχείρισής του, καθώς και τα πλεονεκτήματα που απορρέουν από τη χρήση του, θα ωθήσουν τους developers στην ανάπτυξη ακόμα πιο εκπληκτικών web εφαρμογών, με δυνατότητες που θα προσεγγίζουν όλο και περισσότερο τα desktop προγράμματα. Αναμενόμενο είναι να υπεισέρχονται και κάποια ζητήματα ασφαλείας αλλά, εδώ που τα λέμε, και πού δεν υπεισέρχονται; Καλοδεχούμενο το WebStorage, λοιπόν!


Τα επεισόδια του deltaCast, μοιάζουν με ταινίες.
Στη διάρκεια -- όχι στην ποιότητα.
deltaCast s02e02
notMe-CryptoBSD
http://deltahacker.gr/category/deltacast

Skill: Intermediate
Tags: deterministic, non-deterministic, PRNG, multiplicative congruential method, radioactive decay, thermal noise, cosmic microwave background

ΤΥΧΑΙΟΙ ΑΡΙΘΜΟΙ

μέρος 2ο

by George of the Jungle

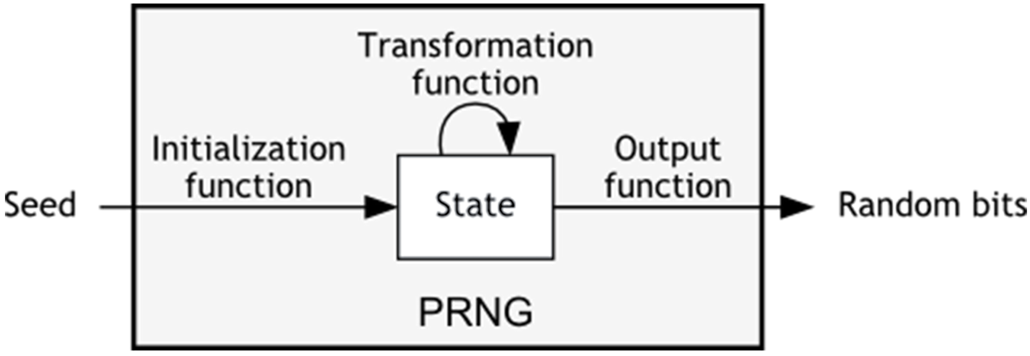
Γεννήτριες τυχαιότητας

Οι τυχαίοι αριθμοί έχουν τεράστια χρησιμότητα στον κόσμο των υπολογιστών. Ένα σύνολο αριθμών, όμως, μπορεί να μοιάζει τυχαίο αλλά στην πραγματικότητα να μην είναι. Μόνο ορισμένα φυσικά φαινόμενα, τα οποία μας είναι αδύνατο να μοντελοποιήσουμε, θα μπορούσαν να βοηθήσουν στη δημιουργία απρόβλεπτων αριθμών και μόνον αυτούς θα μπορούσαμε να θεωρήσουμε πραγματικά τυχαίους. Γι' αυτούς τους λόγους, η κατασκευή ψευδοτυχαίων αριθμών που είναι δύσκολα προβλέψιμοι συγκεντρώνει μεγάλο ενδιαφέρον.

Φανταστείτε έναν περίπλοκο μηχανισμό, ο οποίος λειτουργεί με συγκεκριμένο και γνωστό τρόπο. Ασχέτως του πλήθους των γραναζιών και των ελατηρίων που τον απαρτίζουν, η κατάστασή του είναι πάντοτε προβλέψιμη. Μπορεί να απαιτούνται αρκετοί και σύνθετοι υπολογισμοί, όπως επίσης και καλή γνώση της αρχικής κατάστασης του μηχανισμού, αλλά τουλάχιστον μπορούμε να είμαστε σίγουροι για την πρόβλεψή μας. Κάτι αντίστοιχο ισχύει και για τις συναρτήσεις που παράγουν ψευδοτυχαίους αριθμούς. Μπορεί ο μηχανισμός τους να είναι περίπλοκος, αλλά η συμπεριφορά τους είναι πάντοτε προβλέψιμη. Ακριβώς γι' αυτό, μια τέτοια "πηγή" αριθμών χαρακτηρίζεται κι ως ντετερμινιστική (deterministic). Φανταστείτε τώρα ένα φυσικό φαινόμενο, το οποίο είτε δεν έχουμε καταφέρει να ερμηνεύσουμε ακόμα, είτε μας είναι δύσκολο να μοντελοποιήσουμε. Ένα τέτοιο φαινόμενο θα μπορούσε να χρησιμεύσει για την παραγωγή αριθμών που είναι αδύνατο να προβλέψουμε κι ως εκ τούτου μπορούμε να τους θεωρούμε *πραγματικά* τυχαίους. Μία τέτοια πηγή αριθμών ονομάζεται μη-ντετερμινιστική (non-deterministic). Σε αυτό το άρθρο θα εξετάσουμε ένα αντιπροσωπευτικό δείγμα από το κάθε είδος.

Γεννήτριες ψευδοτυχαίων αριθμών

Εδώ ανήκουν οι αλγόριθμοι που παράγουν δύσκολα προβλέψιμες αλληλουχίες αριθμών. Βέβαια, οι αλγόριθμοι αυτής της κατηγορίας είναι περισσότερο γνωστοί με το αγγλικό τους όνομα: Pseudo Random Number Generator ή, πιο σύντομα, PRNG. Σε αυτούς τους μηχανισμούς δίνουμε μια τιμή που ονομάζεται seed (σπόρος) κι αυτοί με τη σειρά τους επιστρέφουν μια αλληλουχία ψευδοτυχαίων αριθμών. Αυτό που αξίζει να σημειώσουμε, είναι ότι αν δώσουμε τον ίδιο "σπόρο", θα πάρουμε την ίδια ακριβώς αλληλουχία αριθμών. Συνεπώς, οι εν λόγω αλγόριθμοι μπορούν να δημιουργήσουν τόσες διαφορετικές αλληλουχίες αριθμών, όσες είναι και οι διαφορετικές τιμές που μπορούν να δεχτούν στην είσοδό τους. Κατά παράδοση, το μέγεθος που δίνεται σαν seed σε αυτούς τους αλγόριθμους είναι η τιμή του system clock. Βλέπετε, το συγκεκριμένο "ρολόι" έχει πολύ μεγάλη ακρίβεια και κάθε φορά που το "διαβάζουμε" είναι σχεδόν σίγουρο ότι θα μας δώσει μια διαφορετική τιμή.



Μια γεννήτρια ψευδοτυχαίων αριθμών μπορεί να παράγει τόσες διαφορετικές ακολουθίες, όσα είναι και τα διαφορετικά seeds που μπορεί να δεχτεί. Αν της προσφέρουμε σαν είσοδο το ίδιο seed, θα μας επιστρέφει συνεχώς την ίδια ακολουθία αριθμών.

Παρακάτω βλέπουμε ένα μικρό πρόγραμμα σε C, το οποίο αξιοποιεί έναν μετρητή υψηλής ακρίβειας (high resolution timer) για να μας δώσει ένα seed. Το πρόγραμμα είναι γραμμένο για την πλατφόρμα των Windows και μετράει το χρόνο εκτέλεσης της συνάρτησης sleep(), για μια συγκεκριμένη παράμετρο. Επειδή η εν λόγω συνάρτηση δεν είναι ιδιαίτερα ακριβής στα Windows, ενώ αντίθετα ο μετρητής που χρησιμοποιούμε έχει τεράστια ακρίβεια, το αποτέλεσμα που λαμβάνουμε είναι πολύ συχνά διαφορετικό…

```
#include <windows.h>
#include <iostream.h>
#include <cstdlib.h>

using namespace std;
int main() {
    LARGE_INTEGER freq;
    LARGE_INTEGER start;
    LARGE_INTEGER stop;
    if( QueryPerformanceFrequency( &freq ) ) {
        QueryPerformanceCounter( &start );
        Sleep( 1000 ) ;
        QueryPerformanceCounter( &stop );
        long long time_ms = (stop.QuadPart - start.QuadPart) * 1000 /
                             freq.QuadPart;

        cout << time_ms << "ms" << endl;
    }
    system("PAUSE");
    return 0;
}
```

Φαινομενικά τυχαία διαδοχή

Υπάρχουν πολλοί αλγόριθμοι που παράγουν ψευδοτυχαίους αριθμούς και, όπως υποψιάζεστε, δεν έχουν όλοι την ίδια "ποιότητα". Κάποιοι παρουσιάζουν μεγάλη περιοδικότητα και κάποιοι άλλοι μικρή. Βέβαια, η περιοδικότητα δεν καθορίζεται μόνον από τον αλγόριθμο αυτόν καθ' αυτόν, αλλά κι από διάφορες παραμέτρους που ρυθμίζουν τη λειτουργία του. Εμείς θα ασχοληθούμε με έναν δημοφιλή αλγόριθμο που υλοποιεί τη λεγόμενη "Πολλαπλασιαστική Αναλογική Μέθοδο" (Multiplicative Congruential Method). Ας δούμε πώς παράγονται οι ψευδοτυχαίοι αριθμοί με αυτήν τη μέθοδο…

0. Ορίζουμε μια τιμή seed. Όπως αναφέραμε, μια κατάλληλη πηγή για το seed αποτελούν οι διάφοροι μετρητές του συστήματος. Έστω ότι το seed είναι το μέγεθος X0.

1. Υπολογίζουμε την πρώτη ψευδοτυχαία τιμή με τη σχέση: $X_1 = (a*X_0) \bmod m$, όπου τα a και m είναι φυσικοί αριθμοί.

2. Υπολογίζουμε τη δεύτερη ψευδοτυχαία τιμή με τη σχέση: $X_2 = (a*X_1) \bmod m$, όπου τα a και m είναι (οι ίδιοι) φυσικοί αριθμοί.

ν. Υπολογίζουμε τη ν-οστή ψευδοτυχαία τιμή με τη σχέση: $X_\nu = (a*X_{\nu-1}) \bmod m$, όπου τα a και m είναι (οι ίδιοι) φυσικοί αριθμοί.

Εξαιτίας της πράξης modulo (υπόλοιπο διαίρεσης μεταξύ ακεραίων), είναι φανερό ότι κάθε τιμή Xi που υπολογίζουμε ανήκει στο σύνολο {0, 1, ..., m-1}. Μετά από ένα πεπερασμένο πλήθος βημάτων, το οποίο εξαρτάται από τη σχέση που έχουν μεταξύ τους τα a και m, ο αλγόριθμος αρχίζει να επαναλαμβάνεται. Αυτό το πλήθος βημάτων ονομάζεται και περίοδος του αλγορίθμου. Σε κάθε περίπτωση, η περίοδος είναι μικρότερη ή ίση με m, αφού τόσες είναι και οι διαφορετικές τιμές που μπορεί να μας δώσει ο αλγόριθμος.

Από τα παραπάνω συμπεραίνει κανείς ότι για να πάρουμε μια μεγάλη αλληλουχία αριθμών, πρέπει να επιλέξουμε ένα όσο το δυνατόν μεγαλύτερο m, αφού αυτό καθορίζει τη μέγιστη περίοδο. Επιπρόσθετα, πρέπει να επιλέξουμε το κατάλληλο a, αφού αυτό σε συνδυασμό με το m καθορίζουν την περίοδο. Τέλος, για λόγους απόδοσης του αλγορίθμου, οι τιμές των a και m πρέπει να είναι προσαρμοσμένες στην αρχιτεκτονική του εκάστοτε επεξεργαστή. Έτσι, σε ένα σύστημα με 32bit CPU, θα ήταν φρόνιμο να επιλέξουμε τιμές που περιγράφονται το πολύ με 32 bits κι όχι μεγαλύτερες. Σε αυτή την περίπτωση, μια καλή επιλογή για το m είναι το 2^32 - 1, που βρίσκεται ακριβώς πάνω στο όριο.

	Seed=1	Seed=2	Seed=3	Seed=4	Seed=5	Seed=6	Seed=7	Seed=8	Seed=9	Seed=10
x0	1	2	3	4	5	6	7	8	9	10
x1	13	26	39	52	1	14	27	40	53	2
x2	41	18	59	36	13	54	31	8	49	26
x3	21	42	63	20	41	62	19	40	61	18
x4	17	34	51	4	21	38	55	8	25	42
x5	29	58	23	52	17	46	11	40	5	34
x6	57	50	43	36	29	22	15	8	1	58
x7	37	10	47	20	57	30	3	40	13	50
x8	33	2	35	4	37	6	39	8	41	10
x9	45		7		33		59		21	2
x10	9		27		45		63		17	
x11	53		31		9		51		29	
x12	49		19		53		23		57	
x13	61		55		49		43		37	
x14	25		11		61		47		33	
x15	5		15		25		35		45	
x16	1		3		5		7		9	

Στο πείραμά μας, όταν η τιμή που δίνουμε σαν seed είναι περιττή, η περίοδος της γεννήτριας είναι 16. Μετά από τον 16ο αριθμό, επαναλαμβάνεται από την αρχή.

Έλεγχος επαναληψιμότητας

Επειδή η συνεχόμενη θεωρία σίγουρα κουράζει, αποφασίσαμε να κάνουμε και μερικές δοκιμές: Ορίσαμε το a=13 και το m=64 κι έτσι συγκροτήσαμε τη γεννήτρια $X_n=(13*X_{n-1})$ mod 64. Παίζοντας, διαπιστώσαμε ότι η τιμή του seed (δηλαδή του X_0) επιδρούσε δραματικά! Επιλέγοντας μια περιττή τιμή για το seed, η περίοδος της γεννήτριας γινόταν ίση με 16. Δηλαδή η αλληλουχία που προέκυπτε επαναλαμβανόταν ανά 16 αριθμούς. Αποδίδοντας στο seed μια άρτια τιμή η κατάσταση χειροτέρευε, αφού η περίοδος γινόταν μικρότερη ή ίση με 8. Όπως αντιλαμβάνεστε, η επιλογή των συγκεκριμένων τιμών για τα a και m ήταν κακή: Ο αλγόριθμος παρουσίαζε αρκετά μικρή περίοδο, η οποία μάλιστα παρουσίαζε μεγάλη ευαισθησία στο seed.

Σύμφωνα με τη θεωρία, για να έχουμε τα καλύτερα δυνατά αποτελέσματα το m πρέπει να είναι μεγάλο, ενώ το a πρέπει να *μην* αποτελεί πολλαπλάσιο κάποιου από τους πρώτους παράγοντες του m. Τώρα, αν ο m είναι πρώτος, είναι προφανές ότι έχει μόνον έναν πρώτο παράγοντα: τον εαυτό του. Έτσι, η επιλογή του a γίνεται ευκολότερη. Για παράδειγμα, αν ορίσουμε το a = 762939453125 και το m = 43, η περίοδος της γεννήτριας θα γίνει υπερβολικά μεγάλη. Ακόμη και μια CPU που μπορεί να υπολογίζει πολλές τιμές ανά δευτερόλεπτο (Σ.τ.Ε. Κοντά στα δυόμισι γκατζίλιονς, υποθέτω), θα χρειαζόταν αρκετά χρόνια για να ολοκληρώσει μια περίοδο της γεννήτριας. Όμως μ' αυτά και με τ' άλλα, γυρίσαμε πάλι στη θεωρία. Αν θέλετε να δοκιμάσετε τα όρια του δικού σας συστήματος, μπορείτε ν' ανοίξετε ένα πρόγραμμα spreadsheet (όπως το Excel) και να υπολογίσετε τις τιμές της ακόλουθης γεννήτριας:

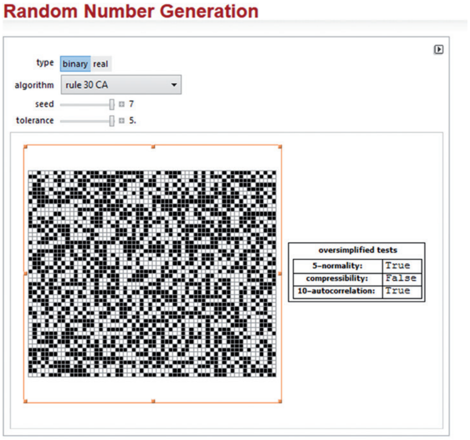
X_n = (762939453125 * X_{n-1}) mod 43, για X_0 = 1

Χαζεύοντας τα αποτελέσματα του συγκεκριμένου πειράματος, είναι σίγουρο ότι το μέγεθος της περιόδου της γεννήτριας θα σας εντυπωσιάσει. Αλήθεια, τι θα λέγατε αν σας έδιναν ένα μέρος αυτών των αριθμών και σας ζητούσαν να αποφανθείτε για το αν είναι τυχαίοι ή όχι; Με μια γρήγορη ματιά στους αριθμούς, είναι δύσκολο να απαντήσουμε σε αυτό το ερώτημα. Θα πρέπει προηγουμένως να χρησιμοποιήσουμε ειδικές συναρτήσεις, οι οποίες ελέγχουν την κατανομή των αριθμών. Σε αυτό το σημείο, αξίζει να αναφέρουμε ο σχετικός έλεγχος μπορεί να γίνει και με έναν πολύ πιο απλό τρόπο. Ο ανθρώπινος εγκέφαλος διαθέτει έναν έμφυτο αλγόριθμο, με τη βοήθεια του οποίου μπορεί να εντοπίζει επαναλαμβανόμενα μοτίβα μέσα σε μια εικόνα. Έτσι, αν οπτικοποιήσουμε τη δοσμένη ακολουθία αριθμών, ενδέχεται να διαπιστώσουμε αμέσως το αν παρουσιάζει περιοδικότητα ή όχι. Φυσικά, αυτή η μέθοδος δεν είναι αλάνθαστη. Υπάρχουν αλγόριθμοι που αν οπτικοποιήσουμε την έξοδό τους θα μπορούμε με δυσκολία να αντιληφθούμε την ύπαρξη περιοδικότητας (αλλά αυτή θα υπάρχει). Αν θέλετε να πραγματοποιήσετε μερικές σχετικές δοκιμές, μπορείτε να χρησιμοποιήσετε το δωρεάν εργαλείο του Steven Wolfram (δημιουργός του Mathematica), στη διεύθυνση

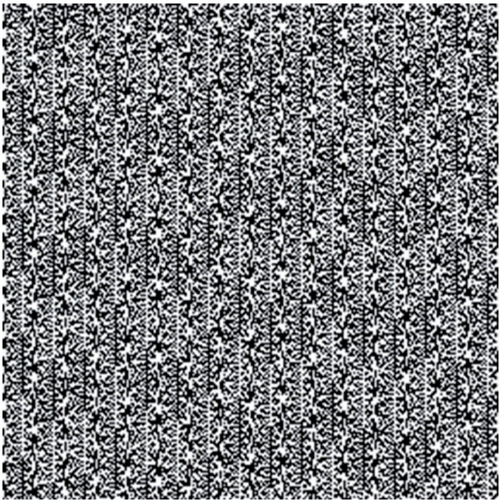
http://demonstrations.wolfram.com/RandomNumberGeneration

Πανάκριβη τυχαιότητα

Μια γεννήτρια πραγματικά τυχαίων αριθμών δεν μπορεί παρά να βασίζεται σε φυσικά φαινόμενα. Σε διαδικασίες, δηλαδή, που δεν μπορούμε να μοντελοποιήσουμε και να περιγράψουμε με κάποιον μαθηματικό τρόπο, επομένως είναι αδύνατο να προβλέψουμε την εξέλιξή τους. Οι αριθμοί που παράγουν αυτές οι γεννήτριες είναι "εντελώς τυχαίοι", "απαρέγκλιτα τυχαίοι", "απόλυτα τυχαίοι" και πάει λέγοντας. (Εντάξει, μ' αυτές τις εκφράσεις θέλαμε μόνο να τσιγκλήσουμε την κοινή αντίληψη περί τύχης :))



Εδώ έχουμε οπτικοποιήσει μια ψευδοτυχαία ακολουθία, που δημιουργήθηκε με τον αλγόριθμο "rule 30 CA" του Mathematica. Από την εικόνα διαπιστώνουμε ότι δεν υπάρχει κάποιο επαναλαμβανόμενο μοτίβο και συμπεραίνουμε ότι οι αριθμοί δεν παρουσιάζουν περιοδικότητα. Στο ίδιο συμπέρασμα θα καταλήξουμε κι αν καταφύγουμε σε στατιστικά test, όπως το "10 – autocorrelation".



Εδώ έχουμε οπτικοποιήσει τους αριθμούς που παράγει η συνάρτηση rand() της PHP. Όπως βλέπετε, η ύπαρξη επαναλαμβανόμενων μοτίβων είναι ολοφάνερη. Δεν χρειαζόμαστε καμία ειδική συνάρτηση, προκειμένου να διαπιστώσουμε ότι οι αριθμοί που παράγει η rand() είναι ψευδοτυχαίοι.

Οι γεννήτριες πραγματικά τυχαίων αριθμών παρουσιάζουν εξαιρετικά μεγάλη περίοδο, που σε πολλές περιπτώσεις ανέρχεται σε εκατοντάδες χρόνια! Μία τόσο μεγάλη περίοδος εξασφαλίζει ότι είναι *αδύνατο* να προβλέψει κανείς τον επόμενο αριθμό. Γι' αυτόν ακριβώς το λόγο, οι αριθμοί που παράγονται από τέτοιες γεννήτριες είναι ιδιαίτερα χρήσιμοι στην κρυπτογραφία και συγκεκριμένα στη διαδικασία παραγωγής κλειδιών.

Τα φαινόμενα που αξιοποιούνται συνήθως για την παραγωγή τυχαίων αριθμών είναι η ραδιενεργός διάσπαση (radioactive decay), ο θερμικός θόρυβος (thermal noise), η κοσμική ακτινοβολία υποβάθρου (Cosmic Microwave Background) αλλά και τα γνωστά "χιόνια" της αναλογικής τηλεόρασης. Τα παραπάνω φαινόμενα, με εξαίρεση το τελευταίο, αποτελούν τις κατεξοχήν πηγές τυχαιότητας. Όπως καταλαβαίνετε, όμως, οι αντίστοιχες συσκευές είναι ακατάλληλες για χρήση εκτός εργαστηρίου. Ο μετρητής Κοσμικής Ακτινοβολίας, για παράδειγμα, δεν είναι ένα gadget που μπορούμε να κουβαλάμε στην τσέπη μας — ούτε καν στο φορτηγό, δηλαδή. Κάτι αντίστοιχο ισχύει και για τις συσκευές που στηρίζονται στη διάσπαση του ραδιενεργού Κοβαλτίου. Είναι αδύνατο να κουβαλήσουμε μια τέτοια συσκευή, εξαιτίας κυρίως της "άβολης" ραδιενέργειας.

Αυτή τη στιγμή κυκλοφορούν αρκετές συσκευές που αξιοποιούν φυσικά φαινόμενα για την παραγωγή τυχαίων αριθμών. Και δεν αναφερόμαστε σε συσκευές σαν τις παραπάνω, αλλά σε μικρότερες, απόλυτα ασφαλείς και σίγουρα πιο εύχρηστες. Πρόκειται για συσκευές που υποστηρίζουν το δίαυλο PCI Express και άρα μπορούν να προστεθούν σε οποιοδήποτε σύγχρονο υπολογιστή. Οι εν λόγω γεννήτριες εκμεταλλεύονται μια διαδικασία από το πεδίο της κβαντικής οπτικής. Μάλιστα, με τη βοήθεια των βιβλιοθηκών που τις συνοδεύουν, αποτελούν μια εξαιρετικά ποιοτική

πηγή τυχαίων αριθμών, ακόμα και για τα δεδομένα μιας πολυεθνικής επιχείρησης που θέλει να προστατέψει τα μεγαλύτερα μυστικά της. Το μοναδικό μειονέκτημα αυτών των συσκευών έγκειται στο μεγάλο κόστος – τουλάχιστον για τα δεδομένα της δικής μας τσέπης (κι όχι των πολυεθνικών). Και για να 'χετε μια αίσθηση για τι πράγμα μιλάμε, αρκεί να πούμε ότι μία τέτοια κάρτα επέκτασης για τον υπολογιστή κοστίζει περί τα 2500 ευρώ. Τελικά, ακόμη κι αν ξεχάσουμε οτιδήποτε αναφέραμε στη θεωρία, κρίνοντας από το κόστος και μόνο είναι φανερό ότι οι τυχαίοι αριθμοί έχουν τεράστια αξία.



Η συσκευή της εικόνας αξιοποιεί μια διαδικασία από το πεδίο της κβαντικής οπτικής και δημιουργεί τυχαίους αριθμούς. Μπορεί να μην μας είναι απαραίτητη, αλλά αν δεν κόστιζε 2500 ευρώ, θα σκεφτόμασταν πολύ σοβαρά να την αγοράσουμε :)

Αληθινή τύχη, στη φθήνια!

Κλείνοντας, θα ήταν άστοχο από μέρους μας να μη σταθούμε σε μια φτηνή και προσιτή πηγή τυχαίων αριθμών. Αναφερόμαστε στον ηλεκτρομαγνητικό θόρυβο που συλλαμβάνει μια κεραία. Πιο απλά, εννοούμε τα παράσιτα της αναλογικής τηλεόρασης ή, τέλος πάντων, τα "χιόνια" που αναφέραμε προηγουμένως. Πως θα σας φαινόταν αν βάζαμε μια web camera μπροστά από μια αναλογική τηλεόραση και βγάζαμε μια φωτογραφία, κάθε φορά που χρειαζόμαστε έναν τυχαίο αριθμό; Η εικόνα με τα παράσιτα θα μπορούσε εύκολα να μετατραπεί σ' ένα bit stream, αντικαθιστώντας τις λευκές κουκίδες με άσσους και τις μαύρες με μηδενικά. Οι τιμές που θα προέκυπταν μ' αυτόν τον τρόπο θα περνούσαν άνετα οποιοδήποτε στατιστικό test τυχαιότητας.



Μια καλή και φτηνή πηγή τυχαιότητας αποτελεί ο ηλεκτρομαγνητικός θόρυβος που συλλέγει η κεραία της τηλεόρασης. Για να τον εκμεταλλευτούμε, αρκεί να φωτογραφήσουμε την εικόνα με τα παράσιτα και να μετατρέψουμε τις μαύρες κουκίδες σε μηδενικά και τις λευκές σε άσσους!

Ακριβώς αυτή την ιδέα υλοποίησε και ένας συνάδελφος Πληροφορικός, χρησιμοποιώντας ένα Raspberry Pi. Μπορείτε να δείτε ολόκληρο το εγχείρημα στο YouTube: `http://bit.ly/RPiRNGwebcam`. Νομίζουμε ότι αποτελεί μια πραγματικά απλή και όμορφη ιδέα. Και κάπου εδώ, με αυτή την αληθινή χακεριά, θα σας αφήσουμε. Ελπίζουμε να απολαύσατε αυτόν τον περίπατο στον μυστηριώδη κόσμο της τυχαιότητας, όσο κι εμείς!

COOKIES


Η ΠΑΡΑΚΟΛΟΥΘΗΣΗ ΠΟΛΙΤΩΝ ΜΕ ΤΗ ΧΡΗΣΗ ΥΨΗΛΗΣ ΤΕΧΝΟΛΟΓΙΑΣ...
ΕΙΝΑΪ ΕΝΑ ΑΠΑΡΑΔΕΚΤΟ ΦΑΙΝΟΜΕΝΟ...
...ΠΟΥ ΠΡΕΠΕΙ ΝΑ ΣΤΑΜΑΤΗΣΕΙ! ΟΙ ΜΥΣΤΙΚΕΣ ΥΠΗΡΕΣΙΕΣ...
...ΟΦΕΙΛΟΥΝ ΝΑ ΔΡΟΥΝ ΜΕ ΣΕΒΑΣΜΟ ΣΤΑ ΙΔΑΝΙΚΑ...
...ΠΑΝΩ ΣΤΑ ΟΠΟΙΑ ΘΕΜΕΛΙΩΘΗΚΕ ΚΑΙ ΓΙΓΑΝΤΩΘΗΚΕ Η ΥΠΕΡΟΧΗ ΧΩΡΑ ΜΑΣ!
ΑΝΟΙΞΕ ΤΟ ΤΑΜΠΛΕΤ ΜΕ ΑΡΓΕΣ ΚΙΝΗΣΕΙΣ ΚΑΙ ΔΕΙΞΕ ΜΑΣ ΤΑ MAIL ΣΟΥ!..
...ΜΑΔΑΦΑΚΑ!
ΠΑΝΤΣ
SWAT
SWAT
ΠΑΝΟΣ

deltahacker.gr